

普通高等教育“十五”国家级规划教材

# 软 件 工 程

曾强聪

高等教育出版社

# 内容提要

本书是普通高等教育“十五”国家级规划教材。具有结构严谨、概念清晰、内容紧凑，深入浅出、突出实用、便于自学等特点。

全书内容紧凑，深入浅出。全书共 10 章正文，以软件生命周期为主线，主要内容包括：软件工程概论、软件工程过程模型、项目分析与规划、软件需求分析、软件概要设计、面向对象分析与设计、用户界面设计、程序算法设计与编码、软件测试、软件维护。附录部分包括软件文档管理规范 and 软件文档格式。

本书适合于高等应用型本科院校、高等职业学校、高等专科学校、成人高校、本科院校举办的二级职业技术学院使用，也可供示范性软件职业技术学院、继续教育学院、民办高校、技能型紧缺人才培养使用，还可供本科院校、计算机专业人员和爱好者参考使用，并可用作软件技术人员资格（水平）考试的培训教材。

## 图书在版编目(CIP)数据

软件工程 / 曾强聪. —北京：高等教育出版社，  
2004. 11  
ISBN 7-04-015743-8

. 软... . 曾... . 软件工程－高等学校－  
教材 . TP311.5

中国版本图书馆 CIP 数据核字（2004）第 105933 号

策划编辑 冯 英      责任编辑 严 亮      封面设计 王凌波      责任绘图 黄建英  
版式设计 胡志萍      责任校对 王效珍      责任印制

出版发行	高等教育出版社	购书热线	010-64054588
社 址	北京市西城区德外大街 4 号	免费咨询	800-810-0598
邮政编码	100011	网 址	<a href="http://www.hep.edu.cn">http:// www.hep.edu.cn</a>
总 机	010-58581000		<a href="http://www.hep.com.cn">http:// www.hep.com.cn</a>

经 销 新华书店北京发行所  
印 刷

开 本	787 × 1092 1/16	版 次	年 月第 1 版
印 张	14.75	印 次	年 月第 次印刷
字 数	360 000	定 价	18.80 元

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

版权所有 侵权必究

物料号：15743-00

# 出版说明

为加强高职高专教育的教材建设工作,2000 年教育部高等教育司颁发了《关于加强高职高专教育教材建设的若干意见》(教高司[2000]19 号),提出了“力争经过 5 年的努力,编写、出版 500 本左右高职高专教育规划教材”的目标,并将高职高专教育规划教材的建设工作分为两步实施:先用 2 至 3 年时间,在继承原有教材建设成果的基础上,充分汲取近年来高职高专院校在探索培养高等技术应用性专门人才和教材建设方面取得的成功经验,解决好高职高专教育教材的有无问题;然后,再用 2 至 3 年的时间,在实施《新世纪高职高专教育人才培养模式和教学内容体系改革与建设项目计划》立项研究的基础上,推出一批特色鲜明的高质量的高职高专教育教材。根据这一精神,有关院校和出版社从 2000 年秋季开始,积极组织编写和出版了一批“教育部高职高专规划教材”。这些高职高专规划教材是依据 1999 年教育部组织制定的《高职高专教育基础课程教学基本要求》(草案)和《高职高专教育专业人才培养目标及规格》(草案)编写的,随着这些教材的陆续出版,基本上解决了高职高专教材的有无问题,完成了教育部高职高专规划教材建设工作的第一步。

2002 年教育部确定了普通高等教育“十五”国家级教材规划选题,将高职高专教育规划教材纳入其中。“十五”国家级规划教材的建设将以“实施精品战略,抓好重点规划”为指导方针,重点抓好公共基础课、专业基础课和专业主干课教材的建设,特别要注意选择一部分原来基础较好的优秀教材进行修订使其逐步形成精品教材;同时还要扩大教材品种,实现教材系列配套,并处理好教材的统一性与多样化、基本教材与辅助教材、文字教材与软件教材的关系,在此基础上形成特色鲜明、一纲多本、优化配套的高职高专教育教材体系。

普通高等教育“十五”国家级规划教材(高职高专教育)适用于高等职业学校、高等专科学校、成人高校及本科院校举办的二级职业技术学院、继续教育学院和民办高校使用。

教育部高等教育司

2002 年 11 月 30 日

# 前 言

软件工程诞生于 20 世纪 70 年代，现已发展成为计算机领域的一个专门学科。而在我国，从 20 世纪 80 年代末期以来，诸多的软件企业也逐步地由早期的软件作坊开发模式，进入到软件工程开发模式。可以说，最近十几年以来，软件工程作为一门有关软件开发的工程方法学，直接影响到了我们国家的软件产业的发展，它在软件开发中的指导意义与基础地位已经越来越多地得到了整个信息业界的高度重视。

软件工程在软件产业中的基础地位直接决定着它在教学上必将受到的重视程度。软件工程已经成为计算机及其相关专业的专业核心课程，许多有关软件技术的职业认证考试或等级考试也把软件工程列入考试大纲，而且所占考分比例逐年增大。

本书为国家十五规划教材，作者自感责任重大，不敢有半点疏忽，收集资料、潜心论证、精心创作，经过整整两年的精心创作与教学试验，终于有了现在这几十万字的收获。本书完稿之后，殷志云教授（留法博士）在百忙之中抽时间审阅了全部书稿，在此特表示感谢！

本书主要为高等院校应用型本科、专科和高职院校计算机相关专业“软件工程”课程教学所写，其以系统性、科学性、实用性为原则，并以结构严谨、布局合理、概念清晰、内容适度，能够更好地适应“软件工程”课程教学的需要为创作目标。

全书由正文和附录两个部分组成。正文章节包含概述、过程模型、项目分析与规划、需求分析、概要设计、面向对象分析与设计、界面设计、算法设计与编码、测试、维护等内容，它们基本上是按照软件生命周期的顺序进行组织。另外，两个附录则分别给出了软件开发中必然涉及的文档管理规范与文档格式这两个方面的说明。

本书是作者多年来从事软件工程教学、进行软件工程研究的结晶，教材所采用的以软件生命周期为线索实施教学的策略，能够使软件工程教学与软件工程实践得到更有成效的结合。一些正处于发展中的工程方法学，如组件技术、对象分布式技术、UML 建模技术以及软件工程文化等，都在书中得到了体现。

本书也是对作者多年来进行软件工程实践的一次经验总结。教材中的诸多软件问题或工程实例，有许多就取材于作者的软件开发实践，并都按照教学的需要进行了模型简化。显然，这些源于实践的工程问题，对于提高软件工程教学的实践性与实用性，将具有很好的示范效应。为了方便读者自学，教材提供了比较丰富的实例与习题，并且每章都有小结。

本书以服务教学与广大读者为宗旨，但由于作者水平有限，书中难免有不足之处，恳请广大读者批评指正，以便本书再版时不断修正与完善。

曾强聪  
2004 年 9 月

# 目 录

第 1 章 软件工程概述 .....	1	2.5 螺旋模型 .....	22
1.1 软件 .....	1	2.6 喷泉模型 .....	24
1.1.1 软件特点 .....	1	2.7 组件复用模型 .....	24
1.1.2 软件分类 .....	1	小结 .....	25
1.1.3 软件发展历程 .....	3	习题 .....	26
1.2 软件危机 .....	4	第 3 章 项目分析与规划 .....	27
1.2.1 软件危机现象 .....	4	3.1 计算机系统分析 .....	27
1.2.2 产生软件危机的原因 .....	5	3.1.1 计算机系统 .....	27
1.3 软件工程 .....	6	3.1.2 系统分析方法 .....	28
1.3.1 软件工程概念 .....	6	3.1.3 建立系统模型 .....	28
1.3.2 软件工程技术 .....	6	3.2 项目可行性分析 .....	30
1.3.3 软件工程管理 .....	9	3.2.1 可行性分析意义 .....	30
1.3.4 软件工程基本原则 .....	10	3.2.2 可行性分析内容 .....	31
1.3.5 软件工程目标 .....	12	3.2.3 可行性分析过程 .....	32
1.3.6 软件工程文化 .....	12	3.3 项目成本效益分析 .....	33
小结 .....	13	3.3.1 项目成本估算 .....	33
习题 .....	14	3.3.2 项目效益分析 .....	35
第 2 章 软件工程过程模型 .....	15	3.4 项目规划 .....	36
2.1 软件生命周期 .....	15	3.4.1 项目开发计划 .....	37
2.1.1 软件定义期 .....	15	3.4.2 项目进度表 .....	37
2.1.2 软件开发期 .....	16	小结 .....	38
2.1.3 软件运行与维护期 .....	17	习题 .....	39
2.2 瀑布模型 .....	17	第 4 章 软件需求分析 .....	41
2.2.1 瀑布模型的特点 .....	18	4.1 需求分析的任务 .....	41
2.2.2 瀑布模型的作用 .....	18	4.1.1 用户需求 .....	41
2.2.3 带有信息反馈环的瀑布模型 .....	18	4.1.2 系统需求 .....	42
2.2.4 瀑布模型的局限 .....	19	4.2 需求分析过程 .....	42
2.3 原型模型 .....	19	4.3 用户需求获取 .....	43
2.3.1 快速原型方法 .....	19	4.3.1 研究用户 .....	43
2.3.2 原型进化模型 .....	20	4.3.2 从调查中获得用户需求 .....	44
2.4 增量模型 .....	21	4.3.3 通过原型完善用户需求 .....	45
2.4.1 增量模型的特点 .....	21	4.3.4 用户需求陈述 .....	45
2.4.2 增量模型的作用 .....	22	4.4 结构化分析建模 .....	47

4.4.1 功能层次模型 .....	47	6.1.3 UML 建模方法 .....	105
4.4.2 数据流模型 (DFD 图) .....	47	6.2 面向对象分析建模 .....	107
4.4.3 数据关系模型 (ER 图) .....	54	6.2.1 用例图 .....	107
4.4.4 系统状态模型 .....	56	6.2.2 活动图 .....	112
4.5 需求有效性验证 .....	58	6.2.3 分析类图 .....	113
4.5.1 需求验证内容 .....	58	6.2.4 序列图 .....	117
4.5.2 需求验证方法 .....	59	6.3 面向对象设计建模 .....	118
4.6 需求规格定义 .....	60	6.3.1 设计类图 .....	119
小结 .....	61	6.3.2 协作图 .....	119
习题 .....	62	6.3.3 状态图 .....	121
第 5 章 软件概要设计 .....	64	6.3.4 构件图 .....	122
5.1 概要设计过程与任务 .....	64	6.3.5 部署图 .....	122
5.1.1 设计过程 .....	64	小结 .....	123
5.1.2 设计任务 .....	65	习题 .....	124
5.2 系统构架设计 .....	67	第 7 章 用户界面设计 .....	126
5.2.1 集中式结构 .....	67	7.1 用户界面设计过程 .....	126
5.2.2 客户机/服务器结构 .....	68	7.2 界面设计中需要考虑的因素 .....	127
5.2.3 多层客户机/服务器结构 .....	69	7.3 界面类型 .....	129
5.2.4 组件对象分布式结构 .....	71	7.3.1 单窗体界面 (SDI) .....	129
5.3 软件结构设计 .....	73	7.3.2 多窗体界面 (MDI) .....	130
5.3.1 模块概念 .....	74	7.3.3 辅助窗体 .....	131
5.3.2 模块的独立性 .....	76	7.3.4 Web 页面 .....	132
5.3.3 结构化设计建模 .....	81	7.4 界面功能特征 .....	133
5.3.4 软件结构优化 .....	84	7.4.1 用户交互 .....	133
5.4 面向数据流的结构设计 .....	87	7.4.2 信息表示 .....	134
5.4.1 变换流分析与设计 .....	87	7.4.3 用户联机支持 .....	135
5.4.2 事务流分析与设计 .....	88	7.5 界面导航设计 .....	136
5.4.3 混合流分析与设计 .....	90	小结 .....	137
5.4.4 设计举例 .....	91	习题 .....	138
5.5 数据库结构设计 .....	92	第 8 章 程序算法设计与编码 .....	139
5.5.1 逻辑结构设计 .....	93	8.1 结构化程序特征 .....	139
5.5.2 物理结构设计 .....	96	8.2 程序算法设计工具 .....	140
小结 .....	97	8.2.1 程序流程图 .....	140
习题 .....	98	8.2.2 N-S 图 .....	141
第 6 章 面向对象分析与设计 .....	101	8.2.3 PAD 图 .....	142
6.1 面向对象方法学 .....	101	8.2.4 PDL 语言 .....	144
6.1.1 面向对象方法的基本概念 .....	101	8.2.5 判定表 .....	145
6.1.2 面向对象方法具有的优越性 .....	104	8.3 Jackson 程序设计方法 .....	146

8.3.1 Jackson 数据结构图	146	10.1 软件维护概述	188
8.3.2 Jackson 程序设计步骤	146	10.1.1 软件维护定义	188
8.3.3 Jackson 程序设计举例	147	10.1.2 影响软件维护工作的因素	189
8.4 程序编码	151	10.1.3 非结构化维护与 结构化维护	189
8.4.1 编程语言种类	151	10.1.4 软件维护的代价	190
8.4.2 选择编程语言的依据	153	10.2 软件可维护性	190
8.4.3 编程风格与质量	154	10.3 软件维护的实施	191
8.4.4 影响程序工作效率的因素	158	10.3.1 维护机构	191
8.5 程序算法复杂性度量	159	10.3.2 维护申请报告	193
小结	161	10.3.3 软件维护工作流程	193
习题	163	10.3.4 维护记录	194
第 9 章 软件测试	165	10.3.5 维护评价	194
9.1 软件测试基本概念	165	10.4 对老化系统的维护	195
9.1.1 测试目标	165	10.5 逆向工程与再工程	195
9.1.2 测试方法	166	10.6 软件配置管理	196
9.1.3 测试中的信息流	166	10.6.1 配置标识	197
9.2 软件测试过程	168	10.6.2 变更控制	197
9.2.1 单元测试	168	10.6.3 版本控制	197
9.2.2 集成测试	170	小结	198
9.2.3 确认测试	173	习题	199
9.3 软件测试用例设计	175	附录 A 软件文档管理规范	200
9.3.1 白盒测试用例设计	175	A.1 软件文档说明	200
9.3.2 黑盒测试用例设计	178	A.1.1 软件文档的定义及作用	200
9.4 面向对象测试	180	A.1.2 软件文档分类	200
9.4.1 面向对象单元测试	180	A.1.3 软件文档与软件生命周期 之间的关系	201
9.4.2 面向对象集成测试	180	A.1.4 文档的使用者	202
9.4.3 面向对象确认测试	180	A.1.5 文档编码规则	202
9.5 软件调试	181	A.2 软件文档格式	203
9.5.1 调试方法	181	A.3 软件文档管理规范	205
9.5.2 调试策略	181	A.4 软件文档的质量评价	207
9.6 自动测试工具	183	附录 B 软件文档格式	208
9.7 软件可靠性评估	184	B.1 可行性研究报告	208
9.7.1 可靠性概念	184	B.2 项目计划说明书	209
9.7.2 估算系统平均无故障时间	184	B.3 需求规格说明书	211
9.7.3 估算系统中的故障总数	185	B.4 概要设计说明书	212
小结	185	B.5 数据库设计说明书	215
习题	186		
第 10 章 软件维护	188		

B.6 详细设计说明书 .....	216	B.11 测试分析报告 .....	223
B.7 模块开发卷宗 .....	217	B.12 系统试运行计划书 .....	224
B.8 用户操作手册 .....	218	B.13 项目开发总结报告 .....	225
B.9 系统维护手册 .....	220	参考文献 .....	227
B.10 测试计划书 .....	222		



# 第 1 章 软件工程概述

在计算机发展早期，软件被等同于程序，开发软件也就是编写程序。但是，随着软件应用的推广与规模的扩大，软件由程序发展成为了软件产品，软件项目成为了软件开发中的工程任务计量单位，软件作为工程化产品则被理解为程序、数据、文档等诸多要素的集合。应该说，软件工程即是为适应软件的产业化发展需要，而逐步发展起来的一门有关软件项目开发的工程方法学。

## 1.1 软 件

### 1.1.1 软件特点

计算机系统由硬件、软件两部分组成。早期的计算机系统以硬件为主，软件费用只占系统总费用的 20%左右。但随着计算机应用范围的扩大，软件需求急剧上升，到 20 世纪 90 年代的时候，软件费用已经上升到了系统总费用的 80%以上。

早期的计算机系统主要用于一些科研机构的科学工程计算，软件任务单一，应用面狭窄，软件问题相对比较简单。因此早期软件开发主要着眼于程序编写，程序成为了早期软件的代名词。但随着软件系统功能的增多，软件应用范围与规模的扩大，软件变得越来越复杂了。因此，今天的软件系统已经具有了更多的内容，包含：计算机程序、用于设置程序正常工作的配置文件、描述软件构造的系统文档、指导用户正确使用软件的用户文档等。

软件是计算机系统逻辑成分，相对于硬件的有形的物理特性，软件则是抽象的，具有无形性。例如程序，它是操纵计算机工作的指令的集合，但它并不能以一种特殊的物理形态独立存在，而必须通过它在它之外的物理存储介质，例如，磁盘、磁带等，才能得以保存；而当需要它工作时，则需要将它调入到计算机的内部存储器上，并且通过计算机的中央处理器才能工作。

软件的无形特性使得我们只能从软件之外去观察、认识软件。可以把计算机系统与人的大脑进行比较，计算机硬件如同人脑的生理构造，而软件则如同基于人脑而产生的人的知识、思想等。

### 1.1.2 软件分类

计算机系统往往需要许多不同种类的软件协同工作，软件工程人员也可能会承担各种不同种类的软件开发、维护任务。因此，可以从多个不同的角度来划分软件的类别。

#### 1. 按软件功能划分

(1) 系统软件：系统软件是计算机系统的必要成分，它跟计算机硬件紧密配合，以使计算机系统的各个部分协调、高效地工作。例如操作系统、数据库管理系统等。

(2) 支撑软件：用于协助用户开发与维护软件系统的一些工具性软件。例如程序编译器、源程序编辑器、错误检测程序、程序资源库等。

(3) 应用软件：用于为最终用户提供数据服务、事务管理或工程计算的软件。例如商业数据处理软件、工程设计制图软件、办公管理自动化软件、多媒体教学软件等。

## 2. 按软件工作方式划分

(1) 实时处理软件：能够及时进行数据采集、反馈和迅速处理数据的软件。其主要用于特殊设备的监控，例如，自动流水线监控程序、飞机自动导航程序。

(2) 分时处理软件：能够把计算机 CPU 工作时间轮流分配给多项数据处理任务的软件。例如，多任务操作系统。

(3) 交互式软件：能够实现人机对话的软件。这类软件往往通过一定的操作界面接收用户给出的信息，并由此进行数据操作；其在时间上没有严格的限定，但在操作上给予了用户很大的灵活性。例如，商业数据处理软件系统中的客户端程序。

(4) 批处理软件：能够把一组作业按照作业输入顺序或作业优先级别进行排队处理，并以成批加工方式处理作业中数据。例如，汇总报表打印程序。

## 3. 按软件规模划分

(1) 微型软件：一个人几天内即可完成的源程序在 500 行语句以内的软件。这种软件主要供个人临时性使用，软件任务单一，操作简单。一般没有必要建立系统文档。

(2) 小型软件：一个人半年之内即可完成的 2 000 行以内的程序。这种软件通常没有与其他软件的数据接口，主要用于用户企业内部专项任务，大多由最终用户自主开发，软件使用周期短，但软件运行过程中产生的数据则可能在今后系统扩充中有一定的价值。考虑到系统今后有扩充的可能性，软件创建过程中应该提供必要的系统文档支持。

(3) 中型软件：由一个项目小组在一年时间内完成的 5 万行源程序以内的软件系统。中型软件在项目实施过程中有了软件人员之间、软件人员与用户之间的通信，软件开发过程中需要进行资源调配。因此，软件开发过程中已需要系统地采用软件工程方法，例如，项目计划、技术手册、用户手册等文档资源，以及技术审查制度等，都需要比较严格地建立起来。

(4) 大型软件：由一个至几个项目小组在两年时间内完成的 5 万行源程序以上的软件系统。当有多个项目小组共同参与软件开发时，需要对参加软件开发的软件工程人员实施二级管理，一般将项目按功能子系统分配到各个项目小组，然后再在项目小组内将具体任务分配到个人。由于项目周期较长，在项目任务完成过程中，人员调整往往不可避免，并会出现对新手的培训和逐步熟悉工作环境的问题。对于这样大规模的软件，采用统一的标准，实行严格的审查是绝对必要的。由于软件的规模庞大以及问题的复杂性，往往在开发的过程中会出现一些事先难以预料的不测事件，对此需要有一定的思想与工作准备。

## 4. 按软件服务对象划分

(1) 通用软件：由软件开发机构开发出来的直接提供给市场的软件。例如通用财务软件、通用字处理软件，杀毒软件等。这类软件通常由软件开发机构自主进行市场调查与市场定位，并由此确定软件规格，大多通过一定的商业渠道进行软件销售。

(2) 定制软件：受某个或少数几个特定客户的委托，由一个或多个软件开发机构在合同的约束下开发出来的软件。例如，某专门设备的控制系统、某特定企业的业务管理系统、某智能

大厦的监控与管理系统、某城市的交通监管系统。这类软件通常由用户进行软件描述，并以此作为基本依据确定软件规格。作为软件开发机构，则必须按照用户要求进行开发。

### 1.1.3 软件发展历程

计算机系统总是离不开软件的，然而早期的硬件、软件是融于一体的，为了使得某台计算机设备能够完成某项工作，不得不给它专门配置程序。但是，随着计算机技术的快速发展和计算机应用领域的迅速拓宽，自 20 世纪 60 年代中期以来，软件需求迅速增长，软件数量急剧膨胀，于是，软件从硬件设备中分离了出来，不仅成为了独立的产品，并且逐渐发展成为了一个专门的产业领域。

观察软件的发展，可以发现软件生产有三个发展时代，即程序设计时代、程序系统时代和软件工程时代。

#### 1. 程序设计时代（20 世纪 50 年代）

20 世纪 50 年代是软件发展的早期时代，计算机主要应用于科研机构的科学工程计算，软件则是为某种特定型号的计算机设备而专门配置的程序。

这时的软件工作者是以个体手工的方式制作软件，他们使用机器语言、汇编语言作为工具直接面对机器编写程序代码。而这时的硬件设备不仅价格昂贵，而且存储容量小、运行速度慢、运行可靠性差。尽管程序要完成的任务在今天看来是简单的，但由于受计算机硬件设备条件的限制，程序设计者也就不得不通过编程技巧来追求程序运行效率。

这个时期的程序大多是自用，程序的编写者往往也就是使用者，软件还没有形成产品。由于早期程序大多是为某个具体应用而专门编写的，程序任务单一，因此，对程序的设计也就仅仅体现在单个程序的算法上。早期程序还往往只能在某台专门的计算机上工作，很难将程序由一台设备移植到另一台设备。

#### 2. 程序系统时代（20 世纪 60 年代）

20 世纪 60 年代，计算机技术迅速发展。在硬件技术上，由于半导体材料、集成电路的出现，计算机设备不仅在运行速度、可靠性和存储容量上有了显著的改善，而且价格也大大降低了，这使得计算机得到了更加广泛的应用。例如，一些大型商业机构已经开始使用计算机进行商业数据处理。

在软件技术上，高级语言的诞生，显著提高了程序编写的效率，这使得一些更大规模的具有综合功能的软件被开发出来。操作系统也出现了，它有效地改善了应用软件的工作环境，并使得应用软件具有了可移植性。由于计算机的应用领域的扩大，软件需求不断增长，软件规模也越来越大；于是，“软件作坊”在这个时期出现了，伴随着“软件作坊”还产生出了具有一定通用性的软件产品。

“软件作坊”已在生产具有工业化特征的软件产品，并且这时的软件工作者已在使用系统的方法设计、制作软件，而不是孤立地对待每个程序。但是，“软件作坊”是一种比较疏散的组织机构，这使得软件开发还不能形成工业流程。

这个时期的软件开发更多地依赖于个人创作。由于软件开发的主要内容仍然是程序编写，软件开发的核心问题仍是技术问题；于是，用户的意图被忽视了，除了程序之外的其他文档、技术标准、软件在今后运行过程中的维护等问题，也往往被忽视了。

软件已经开始成为产品，但软件的生产方式则是跟产品并不适宜的作坊创作方式。于是，随着软件规模的不断扩大，软件危机现象在这个时期最终爆发出来。

### 3. 软件工程时代（20 世纪 70 年代起）

1968 年在联邦德国召开的计算机国际会议上，专门针对软件危机问题进行了讨论，在这次会议上正式提出并使用了“软件工程”术语。于是，软件工程作为一门新兴的工程学科诞生了。“软件工程”如一线霞光，使软件发展步入到了一个新的时代。

在软件开发上，自 20 世纪 70 年代以来的 30 年里，结构化的工程方法获得了广泛应用，并已成为了一种成熟的软件工程方法学；而自 20 世纪 90 年代起，基于面向对象的工程方法，也已被应用于软件开发之中。应该说，采用工程的原理、技术和方法实施软件产品开发，以适应软件产业化发展的需要，成为了这个时期诸多软件企业的追求目标。

这是一个软件产业高速发展的时期，以软件为特征的“智能”产品不断涌现。尤其是网络通讯技术、数据库技术与多媒体技术的结合，彻底改变了软件系统的体系结构，它使得计算机的潜能获得了更大程度的释放。可以说，以计算机软件为核心的信息技术的高速发展，已经使得人们的生活方式与生活节奏发生了根本性的变化。

“软件工程”自产生以来，人们就寄希望于它去冲破“软件危机”这朵乌云。但是，软件危机现象并没有得到彻底排除，特别是，一些老的危机问题可能解决了，但接着又出现了许多新的危机问题，于是不得不去寻找一些更新的工程方法。应该说，正是危机问题的不断出现，推动着软件工程方法学的快速发展。

## 1.2 软件危机

### 1.2.1 软件危机现象

软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。例如，软件的开发成本、进度，软件质量，等等。这些问题绝不仅仅是不能正常运行的软件才具有的，实际上，几乎所有软件都不同程度地存在这些问题。

具体说来，软件危机主要有以下一些方面的典型表现。

#### 1. 软件开发成本、进度的估计很不准确

软件开发机构制定的项目计划跟实际情况有很大差距，使得开发经费一再突破。由于对工作量 and 开发难度估计不足，进度计划无法按时完成，开发时间一再拖延，这种现象严重降低了软件开发机构的信誉。

#### 2. 软件产品常常与用户的要求不一致

在开发过程中，软件开发人员和用户之间缺乏信息交流。开发人员常常是在对用户要求只有模糊了解的情况下就仓促上阵，匆忙着手编写程序。由于这些问题的存在，导致开发出来的软件不能满足用户的实际应用需要。

#### 3. 软件产品质量可靠性差

软件开发过程中，没有建立起确实有效的质量保证体系。一些软件项目为了赶进度或降低软件开发成本，甚至不惜降低软件质量标准、偷工减料。

#### 4. 软件文档不完整、不一致

计算机软件不仅仅是程序，在软件开发过程中还应该产生出一系列的文档资料。实际上，软件开发非常依赖这些文档资料。在软件开发过程中，软件开发机构的管理人员需要使用这些文档资料来管理软件项目；技术人员则需要利用文档资料进行信息交流；用户也需要通过这些文档资料来认识软件，对软件进行验收，熟悉软件的安装、操作等。但是，由于软件项目管理工作的欠缺，软件文档往往不完整，对软件的描述经常不一致，很难通过文档去跟踪软件开发过程中软件产品规格的变更。

#### 5. 软件产品可维护性差

软件中的错误非常难改正，软件很难适应新的硬件环境，很难根据用户的需要在原有软件中增加一些新的功能。这样的软件是不便于重用的，以前开发的软件，一旦过时就不得不完全丢弃。

#### 6. 软件生产率低

软件生产率跟不上硬件的发展速度，不能适应计算机应用的迅速普及，以致现代计算机硬件提供的巨大潜力不能被充分利用。

### 1.2.2 产生软件危机的原因

软件危机现象最初出现在软件发展的第二个阶段——程序系统时代。自那时起，软件工作者就一直在探寻导致软件危机的原因，并期望能通过对软件危机原因的分析，从而找到一种行之有效地克服软件危机的方法、策略。

通过对一系列危机现象的研究，人们总结发现，产生软件危机的原因主要体现在以下几个方面。

#### 1. 软件的不可见特性

软件不同于硬件，它是计算机系统逻辑部件，缺乏“可见性”。硬件错误往往可以通过它的物理现象直接反映出来，例如，出现不正常的发热、噪音现象等；但软件错误没有这些直观表现，例如，软件中存在的程序行错误，就必须等到这行程序执行时才有可能被发现。因此，软件错误比起硬件错误来更难发现。软件的不可见特性也使得对软件项目的量化管理更难实施，对软件质量的量化评价更难操作。

#### 2. 软件系统规模庞大

软件成为产品以后已不同于早期程序，随着它的功能的增多，其规模、复杂程度越来越大。例如，1968年美国航空公司订票系统达到30万条指令；IBM3600S第16版达到100万条指令；1973年美国阿波罗计划达到1000万条指令。这些庞大规模的软件系统，其复杂程度已超过了人所能接受的程度；但是，面对不断复杂的软件系统，其开发手段却仍然需要依靠开发人员的个人创造与手工操作。

#### 3. 软件生产工程化管理程度低

软件生产的工程化管理是软件作为产品所必须的，这意味着软件也需要像硬件一样，在软件分析、设计完成之后，才能考虑软件的实现。应该说，工程化管理能够降低解决问题的代价。但是，许多软件的开发则往往是在分析、设计没有完成的情况下，就已经进入编码实现阶段。由于前期准备工作不充分，致使软件项目管理纷乱，严重影响软件项目成本、开发进度和软件

质量。

#### 4. 对用户需求关心程度不够

软件开发机构不熟悉用户业务领域。软件技术人员所关注的仅仅是计算机技术，它们不太愿意和用户沟通，轻视对用户的需求调查，也缺乏有效的用户调查策略、手段。由于这些问题的存在，使得用户的需求意愿不能充分反映，或被错误理解。

实际上，软件是为用户开发的，只有用户才能真正了解他们自己的需要。由于没有对用户做大量深入细致的调查研究，以致软件需求规格定义不准确，并最终使得完成后的软件不能适应用户的应用需要。

#### 5. 对软件维护重视程度不够

软件开发缺乏统一的规范。在软件产品开发过程中，开发者很少考虑到这个软件今后还需要提供维护。但是，软件的使用周期漫长，软件错误具有隐蔽性，许多年之后软件仍可能需要改错。另外，软件的工作环境也可能会在几年后发生改变；用户也可能在软件运行几年以后，要求对它增加新的功能。这些都是属于软件维护问题。实际上，软件的可维护性是衡量软件质量的一项重要指标，软件可维护性程度高，软件就便于修正、改版和升级，由此可以使软件具有更长的使用寿命。

#### 6. 软件开发工具自动化程度低

尽管软件开发工具比 30 年前已经有了很大的进步，但直到今天，软件开发仍然离不开工程人员的个人创造与手工操作，软件生产仍不可能像硬件设备的生产那样，达到高度的自动化。

## 1.3 软 件 工 程

### 1.3.1 软件工程概念

软件工程是一门关于软件开发与维护的工程学科，它涉及软件生产的各个方面，包括：工程过程、工程原则、技术方法与工具，以及工程项目管理等，能够为经济、高效地开发高质量的软件产品提供最有效的支持。

实际上，我们可以从多个不同的角度来认识软件工程。

1983 年国际权威机构 IEEE 给软件工程下的定义是：软件工程是开发、运行、维护和修复软件的系统方法。其中的“软件”被定义为：计算机程序、方法、规则、相关的文档资料，以及计算机程序运行时所需要的数据。

Fairly 给出的定义是：软件工程学是为了在成本限额以内按时完成开发和修改软件产品时，所需要的系统生产和维护技术及管理学科。

Fritz Baner 则给出了下述定义：软件工程是为了经济地获得可靠的且能在实际机器上有效运行的软件，而建立和使用的完善的工程化原则。

### 1.3.2 软件工程技术

软件工程技术是指软件工程所具有的技术要素。作为软件开发与维护的工程方法学，软件

工程具有三个方面的技术要素，即软件工程方法、软件工具和软件工程过程。

### 1. 软件工程方法

软件工程方法是指完成软件开发与维护任务时，应该“如何做”的技术方法。它所涉及的任务贯穿于软件开发、维护的整个过程之中，包括：软件需求分析、软件结构设计、程序算法设计等诸多任务；而其方法则体现在使用图形或某种特殊语言的方式来表现这些任务中需要建立的软件系统模型，如：数据流模型、软件结构模型、对象模型、组件模型等。主要的软件工程方法有：结构化方法、JSD 方法和面向对象方法。

#### (1) 结构化方法

结构化方法是传统的基于软件生命周期的软件工程方法，自 20 世纪 70 年代产生以来，获得了极有成效的软件项目应用。结构化方法是以软件功能为目标来进行软件构建的，包括：结构化分析、结构化设计、结构化实现和结构化维护等内容，能够很好地适应结构化编程工具，例如：C、Pascal 语言等。它主要使用数据流模型来描述软件的数据加工过程，并可以通过数据流模型，由对软件的分析顺利过渡到对软件的结构设计。

#### (2) JSD 方法

JSD 方法主要用在软件设计上，1983 年由法国科学家 Jackson 提出。它以软件中的数据结构为基本依据来进行软件结构与程序算法设计，是对结构化软件设计方法的有效补充。在以数据处理为主要内容的信息系统开发中，JSD 方法具有比较突出的设计建模优势。

#### (3) 面向对象方法

面向对象方法是以软件问题域中的对象为基本依据来构造软件系统模型的，包括：面向对象分析、面向对象设计、面向对象实现和面向对象维护等内容。确定问题域中的对象成分及其关系，建立软件系统对象模型，是面向对象分析与设计过程中的核心内容。自 20 世纪 80 年代以来，人们提出了许多有关面向对象的方法，其中，由 Booch、Rumbaugh、Jacobson 等人提出的一系列面向对象方法成为了主流方法，并被结合为统一建模语言（UML），成为了面向对象方法中的公认标准。面向对象方法能够最有效地适应面向对象编程工具，例如：C++、Java 等，并特别适用于面向用户的交互式系统的开发。

### 2. 软件工具

软件工具是为了方便软件工程方法的运用而提供的具有自动化特征的软件支撑环境。

软件工具通常也称为 CASE，它是计算机辅助软件工程（Computer-Aided Software Engineering）的英文缩写。CASE 工具覆盖面很广，包括：分析建模、设计建模、源代码编辑生成、软件测试等。表 1-1 所列是一些常用的 CASE 工具类别。

表 1-1 常用的 CASE 工具类别

工 具 类 型	举 例
项目管理工具	项目规划编辑器、用户需求跟踪器、软件版本管理器
软件分析工具	数据字典管理器、分析建模编辑器
软件设计工具	用户界面设计器、软件结构设计器、代码框架生成器
程序处理工具	程序编辑器、程序编译器、程序解释器、程序分析器
软件测试工具	测试数据生成器、源程序调试器

用来支持软件分析、设计的 CASE 工具，如数据字典管理器、分析建模图形编辑器、软件结构设计器等，被称为高端 CASE 工具；而用来支持软件实现和测试的 CASE 工具，如程序编辑器、程序分析器、源程序调试器等，则被称为低端 CASE 工具。

可以把诸多独立的软件工具集成起来，形成一体化的 CASE 工作平台。这样的工作平台能够为软件开发提供更强有力的支持，平台中数据资源共享，界面风格、操作方式统一，诸多通用操作能够作为模块被许多工具调用，一种工具产生的信息也可以被其他的工具引用。

CASE 工作平台最初应用于低端工具的集成上，例如，早期 DOS 环境下的 Turbo C，即是一个集程序管理、编辑、调试、编译于一体的 C 语言程序创建工具。但在今天，CASE 工作平台已被应用于软件开发的各种活动上，涉及软件的分析设计、安装部署、项目管理、版本控制等多个方面。

分析与设计工作平台是软件工程方法中的核心工作平台，由许多工具集成，如图 1-1 所示。分析与设计工作平台主要用于分析、设计阶段的系统建模，许多分析与设计工作平台既可用于结构化方法的系统建模，例如，创建数据流图、软件结构图等；也可用于面向对象方法的系统建模，例如，创建对象图、状态图等。

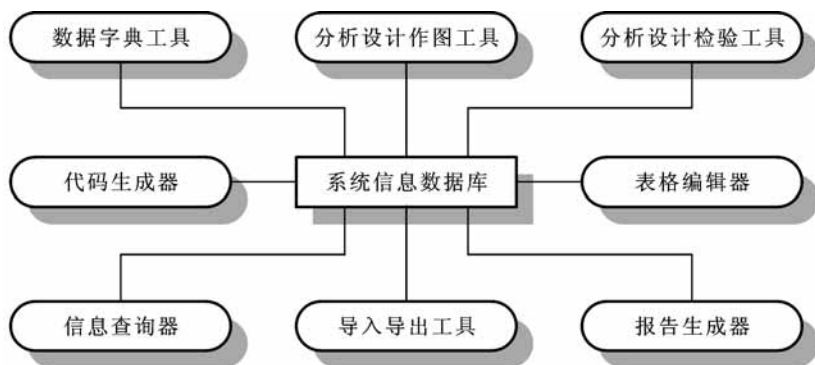


图 1-1 分析与设计工作平台

### 3. 软件工程过程

尽管有软件工具与软件工程方法，但这并不能使软件产品生产完全自动化，它们还需要有合适的软件工程过程才能真正发挥作用。

软件工程过程是指为了开发软件产品，开发机构在软件工具的支持下，按照一定的软件工程方法所进行的一系列软件工程活动。实际上，这一系列的活动也就是软件开发中开发机构需要制定的工作步骤，它应该是科学的、合理的，否则将影响软件开发成本、进度与产品质量。因此，软件工程过程也就涉及到了软件产品开发中有哪些工作步骤，各个工作步骤分别具有什么样的工作特征，以及各个工作步骤分别需要产生一些什么结果等方面的问题。

软件工程过程并不是完全固定的，每个软件开发机构都可以专门制定更加适合自身特点的软件工程过程。实际上，软件产品不同，软件工程过程也可能会有所不同。但是，以下四项基本活动，则是绝大多数软件过程所必须的。



- (1) 软件定义：进行软件规格和使用限制的定义。
- (2) 软件开发：根据软件规格定义制作出软件产品。
- (3) 软件验证：确认软件能够满足用户提出的要求。
- (4) 软件维护：修正软件缺陷，并能根据用户需求变化改进软件。

### 1.3.3 软件工程管理

工程项目总是离不开管理，软件项目也是如此。应该讲，软件项目的管理质量将直接决定软件项目的成败。

软件项目所涉及的管理是多方面的，但概括地看，则主要体现为以下几个方面的管理。

#### 1. 软件项目规划

项目规划就是在项目开始的时候制定出项目开发计划，以明确项目中的人员、任务、进度、费用、文档和目标等，用来指导项目往既定的目标推进。

项目的有效管理直接依赖于项目规划。当面对一个有一定规模的项目时，往往需要针对以下一些问题给出规划。

- (1) 计划项目参与人员的构成、分工与组织方式。
- (2) 对项目所需的硬件、软件资源，以及其他各项费用开支作出估计。
- (3) 进行项目任务分解，明确项目里程碑及其文档成分。
- (4) 制定工作步骤，安排工作进程和人员配备。
- (5) 对项目风险作出估计，并对降低风险给出计划。
- (6) 制定项目监督措施。

#### 2. 项目资源调配

软件项目资源涉及的内容有：

- (1) 硬件设备，如：计算机、打印机、绘图仪等。
- (2) 支撑软件，如：开发工具、数据库系统等。
- (3) 技术资料，如：技术标准手册、参考文献等。
- (4) 项目队伍，如：技术人员、管理人员、协作人员等。

在项目实施的不同阶段对资源的需求是不同的。项目资源调配也就是以项目计划为基本依据，使各个方面资源需求按时到位，并可以根据项目的实际进行情况，对资源作出合理的调整，以保证项目能够按计划顺利进行。

#### 3. 软件产品控制

软件产品控制就是对软件生产过程、软件产品规格等实施有效的控制，并涉及以下两个方面的管理问题。

(1) 软件质量管理。例如，对软件开发中形成的各类文档进行阶段性质量审查，以保证软件开发的规程与标准能够被严格执行；对已经产生的软件产品及其相关文档进行软件评估，以保证开发出来的软件产品与软件的需求规格定义能够保持一致。

(2) 软件配置管理。例如，制定有关软件产品的各项标准，跟踪并记录软件生产过程中发生的变更，标识与存档正在开发的软件的各种不同版本等。

### 1.3.4 软件工程基本原则

软件工程是关于软件项目的工程方法学，其价值只能通过具体的软件项目才能真正体现出来。为保证在软件项目中能够有效地贯彻与正确地使用软件工程规程，需要有一定的软件工程原则来对软件项目加以约束。

一些研究软件工程的专家学者分别从不同的角度陆续提出过许多关于软件工程的原则或“信条”，其中，著名的软件工程专家 B.W.Boehm 经过总结，提出了以下 7 条基本原则。

#### 1. 采用分阶段的生命周期计划，以实现项目的严格管理

软件项目的开展，需要计划在先，实施在后。统计表明，50%以上的失败项目是由于计划不周而造成的。在软件开发、维护的漫长生命周期中，需要完成许多性质各异的工作，假若没有严格有效的计划对项目工作的开展加以约束，则必将使今后在项目中的诸多工作处于一种混乱状态。

采用分阶段的生命周期计划，实现对项目的严格管理，即意味着：应该把软件项目按照软件的生命周期分成若干阶段，以阶段为基本单位制定出切实可行的计划，并严格按照计划对软件的开发、维护实施有效的管理。

#### 2. 坚持阶段评审制度，以确保软件产品质量

软件质量是通过软件产品反映出来的，但是，软件质量的形成将贯穿于整个软件开发过程之中。大量的软件开发事实表明，软件中的许多错误是在开始进行编码之前就已经形成了。根据有关统计：软件设计错误占了软件错误的 63%，而编码错误则仅占 37%。

采用阶段评审制度，也就是要求在软件产品形成过程中，能够对其质量实施过程监控，以确保软件在每个阶段都能够具有较高质量。

实际上，软件错误发现得越早，则错误越容易修正，为此所需付出的代价也就越少。因此，在每个阶段都进行严格的评审，也有利于从管理制度上去减少质量保证成本代价。

#### 3. 实行严格的产品控制，以适应软件规格的变更

软件规格是软件开发与软件验收的基本依据，是不能随意变更的。在软件开发过程中若出现了软件规格的变更，也就意味着，软件的开发费用由此增加了。

但是，在软件开发过程中，改变软件规格有时又是难免的，特别是那些需要较长的开发周期的软件项目。例如，那些由专门用户定制开发的软件系统，就有可能因为用户的业务领域、服务方式发生了改变，而使软件功能有了新的要求。面对用户的新要求显然不能硬性禁止，而只能依靠科学的产品控制技术来适应。实际上，许多通用软件产品也存在规格变更这个问题。例如，软件开发机构为了使自己的软件产品能够在更多的环境下工作，而不得不针对所开发的软件产品推出诸多不同的版本。

实行严格的产品控制，就是当软件规格发生改变时，能够对软件规格进行跟踪记录，以保证有关软件产品的各项配置成分保持一致性，由此能够适应软件规格的变更。

#### 4. 采用先进的程序设计技术

许多先进的软件工程方法往往都起源于先进的程序设计技术。例如，20 世纪 70 年代初出现的 C、Pascal，这些结构化的程序设计语言，不仅成为了当时先进的程序设计技术，而且由此带来了结构化的软件分析、设计方法。自 20 世纪 80 年代以来，随着 C++、Java 等程序设计

语言的产生,面向对象程序设计技术成为了更加先进的技术,并由此推动了面向对象软件分析、设计方法的发展。自 20 世纪 90 年代开始,建立在面向对象程序设计技术基础上的组件技术又随之诞生了,于是,基于组件技术的软件工程方法学也就不断涌现了出来。

采用先进的程序设计技术会获得诸多方面的好处。它不仅会带来更高的软件开发效率,而且所开发出的软件会具有更好的质量,更加便于维护,并且也往往会具有更长久的使用寿命。例如,组件技术,通过创建比起“类”来更加抽象、更具有通用性的基本组件,可以使软件开发如同可插入的零件一样装配。这样的软件,不仅开发容易,维护便利,而且可以根据特定用户的需要,更加方便地进行改装。

#### 5. 软件成果应该能够清楚地审查

软件成果是软件开发的各个阶段产生出来的一系列结果,是对软件开发给出评价的基本依据。包括:系统文档、用户文档、源程序、资源数据和最终产品等内容。

针对软件开发给出有效的评价,是软件工程必须关注的重要内容。但是,软件产品是无形的逻辑产品,缺乏明确的物理度量标准。比起一般物理产品的开发来,软件开发工作进展情况可见性差,其开发更难于管理与评价。因此,为了提高软件开发过程的可见性,更好地管理软件项目和评价软件成果,应该根据软件开发项目的总目标及完成期限,规定软件开发组织的责任和软件成果标准,从而使所得到的结果能够清楚地审查。

#### 6. 开发小组的人员应该少而精

这条基本原则具有以下两点含义。

其一,软件开发小组的组成人员的素质应该好。软件开发是一种需要高度负责、高度协作的高智力劳动。因此,其对人员素质的要求也就主要体现在智力水平、协作能力、团队意识和负责态度等几个方面。实际上,由高素质人员组成的开发队伍能够形成一支很强的开发团队,具有比由一般人员组成的开发队伍高出几倍,甚至几十倍的开发效率,也能够完成更加复杂的项目任务。

其二,软件开发小组的成员人数不宜过多。软件的复杂性和无形性决定了软件开发需要大量的通信。随着软件开发小组人员数目的增加,人员之间因为交流信息、讨论问题而造成的通信开销会急剧增大,这势必影响人员之间的相互协作与工作质量。因此,为了保证开发小组的工作效率,开发小组成员人数一般不应超过 5 人。由于这个原因,一些有许多成员参与的大型项目,也就需要将一个项目分成多个子项目,然后分别交给多个项目小组去完成。

#### 7. 承认不断改进软件工程实践的必要性

软件工程的意义重在实践,作为一门工程方法学,它所推出的一系列原则、方法和标准,不仅来源于工程实践,而且也需要在工程实践中不断地改进、完善。实际上,不同的软件开发机构可以根据自己的具体情况,建立起具有自己特征的软件工程规程体系。

应该讲,上述的六条基本原则,是实现软件开发工程化这个目标的必要前提。但是,仅有上述六条原则,还不足以保证软件开发工程化进程能够持久地进行下去。因此,Boehm 提出了“承认不断改进软件工程实践的必要性”,这表明:软件工程在实际应用中,应该积极主动地采纳新的软件技术,并不断总结新的工程经验。

软件技术在不断进步,软件的应用领域也在不断拓宽。软件工程必须紧紧跟上新时代软件

的发展，才能获得更加持久的生命力。

### 1.3.5 软件工程项目目标

软件工程项目目标是基于软件项目目标的成功实现而提出的，主要体现在以下几个目标上。

- (1) 软件开发成本较低。
- (2) 软件功能能够满足用户的需求。
- (3) 软件性能较好。
- (4) 软件可靠性高。
- (5) 软件易于使用、维护与移植。
- (6) 能按时完成开发任务，并及时交付使用。

应该说，企图让以上几个目标都能够达到理想程度的想法往往是难以实现的。在一个具体项目中，以上几个目标之间很可能会出现冲突，例如，若只顾降低开发成本，则可能会由此导致软件的性能与可靠性也随之降低；另一方面，如果太过于追求软件的性能，则可能会使得开发出来的软件对硬件有较大的依赖，导致软件的可移植性下降。因此，在实际的软件项目中，往往需要针对以上几个目标进行符合实际应用需要的平衡选择。

软件工程的首要问题是软件质量。因此，在涉及平衡软件工程项目目标这个问题的时候，软件的质量应该摆到最重要的位置加以考虑。例如，软件的可用性、有效性、可靠性和可维护性等，它们需要给予特别关注。

### 1.3.6 软件工程文化

早期软件工程主要谋求解决的是技术问题，但是，随着软件技术的进步，软件质量要求的提高和软件产业的发展，软件工程不得不考虑工程中的文化因素。例如，工程人员在软件开发中所应该具备的产品质量观、价值观、道德准则和团队意识等。

软件企业中的工程文化需要逐渐积累。首先是价值观，软件企业上下对软件工程价值需要有一致的认同，假如某个软件企业没有真正认识到软件工程对自己企业发展的影响，则不可能形成自己的软件工程文化。

接着是工程思想与工程行为，软件工程文化是通过一系列具有工程特征的制度、标准展现出来的，例如，工作制度、工程规则、质量标准等。这些东西不仅需要以文件形式印发出来，而且需要被所有员工学习、理解，并成为他们的行为指南。当诸多工程思想被所有员工接受而影响到他们的行为，并由此转化为软件产品的质量与价值之时，软件工程文化的作用也就体现出来了。

观察许多成功的软件企业可以发现，当软件工程在软件企业得到有效应用以后，软件企业中的工程人员会显得更具有工程意识，会更加关心自己的工作质量，也会更加注重同事之间的相互协作。为了提高软件产品的先进性和竞争力，他们会更加积极主动地应用新方法与新技术。

应该说，软件工程文化是企业文化的重要内容之一，它为软件企业中软件工程应用的不断深入创造了非常有利的现场工作环境，随着软件企业中工程经验的积累，这种工程文化氛围会不断地得到加强，如图 1-2 所示。

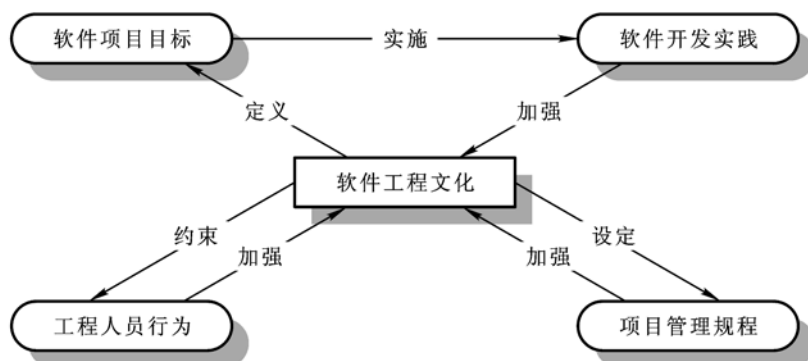


图 1-2 软件工程文化

## 小 结

## 1. 软件特点

软件是计算机系统逻辑成分，具有无形性。其主要内容包括：程序、配置文件、系统文档、用户文档等。

## 2. 软件分类

- (1) 按功能划分：系统软件、支撑软件、应用软件。
- (2) 按工作方式划分：实时处理软件、分时处理软件、交互式软件、批处理软件。
- (3) 按规模划分：微型软件、小型软件、中型软件、大型软件。
- (4) 按服务对象划分：通用软件、定制软件。

## 3. 软件发展阶段

- (1) 程序设计时代（20 世纪 50 年代）。
- (2) 程序系统时代（20 世纪 60 年代）。
- (3) 软件工程时代（20 世纪 70 年代起）。

## 4. 软件危机

(1) 危机现象：开发成本与进度估计不准确，产品与用户要求不一致，产品质量可靠性差，文档不完整不一致，产品可维护性差，生产率低。

(2) 危机原因：软件的不可见性，系统规模庞大，生产工程化程度低，对用户需求关心不够，对维护不够重视，开发工具自动化程度低。

## 5. 软件工程

软件工程是一门关于软件开发与维护的工程学科，它涉及软件生产的各个方面，能够为经济、高效地开发高质量的软件产品提供最有效的支持。

- (1) 工程方法：结构化方法、JSD 方法、面向对象方法。
- (2) 软件工具：具有自动化特征的开发集成支撑环境。
- (3) 工程过程：在软件工具支持下的一系列工程活动，基本活动是软件定义、软件开发、软件验证、软件维护。

(4) 工程管理：项目规划，项目资源调配，软件产品控制。

(5) 工程原则：分阶段生命周期计划，阶段评审制度，严格的产品控制，采用先进的技术，成果能清楚地审查，开发队伍精练，不断改进工程实践。

(6) 工程目标：开发成本较低，软件功能能满足用户需求，软件性能较好，软件可靠性高，软件易于使用、维护与移植，能按时完成开发任务并及时交付使用。

(7) 工程文化：包括工程价值、工程思想和工程行为三个方面的内容。

## 习 题

1. 对于硬件是有形的而软件是无形的观点，有人提出了不同的看法。他认为软件也是有形的，例如，软件需要安装才能工作，软件安装之后会占据一定的磁盘空间。对此，你是什么看法？

2. 软件按服务对象的不同可分为通用软件和定制软件。试举例说明这两类软件的区别。

3. 程序系统时代出现的“软件作坊”有什么特点？

4. 某软件公司抢时间为某单位开发了一个人事管理软件，但软件交付用户使用一段时间之后，用户有了抱怨，原因是单位里某个职工改了名字，但人事管理软件却不允许修改姓名，而只能删除整条记录以后重新输入。试从软件危机角度对这个问题做一些评论。

5. 什么是软件工程？

6. 什么是软件工程方法？简要说明一些主要的软件工程方法。

7. 什么是软件工具？Visual C 是属于什么类型的软件工具？

8. 你是如何看待软件工程过程的？软件过程中最基本的活动有哪些？

9. 软件工程管理主要包括哪些方面的内容？

10. B.W.Boehm 提出的软件工程基本原则的作用是什么？

11. 什么是软件工程目标？如何解决多个目标之间出现的矛盾？

12. 什么是软件工程文化？其中的价值观是什么？

## 第2章 软件工程过程模型

如同任何事物都有一个发生、发展、成熟，直至衰亡的全过程一样，软件系统或软件产品也有一个定义、开发、运行维护，直至被淘汰这样的全过程，我们把软件将要经历的这个全过程称为软件的生命周期。

为了使软件生命周期中的各项任务能够有序地按照规程进行，需要一定的工作模型对各项任务给以规程约束，这样的工作模型被称为软件过程模型，或软件生命周期模型。它是一个有关项目任务的结构框架，规定了软件生命周期内各项任务的执行步骤与目标。

本章将介绍瀑布模型、原型模型、螺旋模型、喷泉模型和组件模型等过程模型。需要注意的是，这些模型并不是有关软件开发进程的固定格式，而只是一种参考标准。实际上，不同的软件项目需要不同的过程模型提供支持，并且还需要根据项目的具体情况，软件开发机构工作方式、管理模式等，对一些标准模型进行适当的调整与补充，以适应项目应用的需要。

### 2.1 软件生命周期

根据我国国家标准《计算机软件开发规范》(GB 8566—8)，软件生命周期包含：软件定义、软件开发、软件运行维护三个时期，并可以细分为可行性研究、项目计划、需求分析、概要设计、详细设计、编码实现与单元测试、系统集成测试、系统确认验证、系统运行与维护等几个阶段。应该说，这是软件生命周期的基本构架，在实际软件项目中，根据所开发软件的规模、种类，软件开发机构的习惯做法，以及软件开发中所采用的技术方法等，可以对各阶段进行必要的合并、分解或补充。

#### 2.1.1 软件定义期

软件定义是软件项目的早期阶段，主要由软件系统分析人员和用户合作，针对有待开发的软件系统进行分析、规划和规格描述，确定软件是什么，为今后的软件开发做准备。这个时期往往需要分阶段地进行以下几项工作。

##### 1. 软件任务立项

软件项目往往开始于任务立项，并需要以“软件任务立项报告”的形式针对项目的名称、性质、目标、意义和规模等作出回答，以此获得对准备着手开发的软件系统的最高层描述。

##### 2. 项目可行性分析

在软件任务立项报告被批准以后，接着需要进行项目可行性分析。

可行性分析是针对准备进行的软件项目进行的可行性风险评估。因此，需要对准备开发的软件系统提出高层模型，并根据高层模型的特征，从技术可行性、经济可行性和操作可行性这三个方面，以“可行性研究报告”的形式，对项目作出是否值得往下进行的回答，由此决定项目是否继续进行下去。

### 3. 制定项目计划

在确定项目可以进行以后,接着需要针对项目的开展,从人员、组织、进度、资金、设备等多个方面进行合理的规划,并以“项目开发计划书”的形式提交书面报告。

### 4. 软件需求分析

软件需求分析是软件规格描述的具体化与细节化,是软件定义时期需要达到的目标。

需求分析要求以用户需求为基本依据,从功能、性能、数据、操作等多个方面,对软件系统给出完整、准确、具体的描述,用于确定软件规格。其结果将以“软件需求规格说明书”的形式提交。

在软件项目进行过程中,需求分析是从软件定义到软件开发的最关键步骤,其结论不仅是今后软件开发的基本依据,同时也是今后用户对软件产品进行验收的基本依据。

## 2.1.2 软件开发期

在对软件规格完成定义以后,接着可以按照“软件需求规格说明书”的要求对软件实施开发,并由此制作出软件产品。这个时期需要分阶段地完成以下几项工作。

### 1. 软件概要设计

概要设计是针对软件系统的结构设计,用于从总体上对软件的构造、接口、全局数据结构和数据环境等给出设计说明,并以“概要设计说明书”的形式提交书面报告,其结果将成为详细设计与系统集成的基本依据。

模块是概要设计时构造软件的基本元素,因此,概要设计中软件也就主要体现在模块的构成与模块接口这两个方面上。结构化设计中的函数、过程,面向对象设计中的类、对象,它们都是模块。概要设计时并不需要说明模块的内部细节,但是需要进行全部的有关它们构造的定义,包括功能特征、数据特征和接口等。

在进行概要设计时,模块的独立性是一个有关质量的重要技术性指标,可以使用模块的内聚、耦合这两个定性参数对模块独立性进行度量。

### 2. 软件详细设计

设计工作的第二步是详细设计,它以概要设计为依据,用于确定软件结构中每个模块的内部细节,为编写程序提供最直接的依据。

详细设计需要从实现每个模块功能的程序算法和模块内部的局部数据结构等细节内容上给出设计说明,并以“详细设计说明书”的形式提交书面报告。

### 3. 编码和单元测试

编码是对软件的实现,一般由程序员完成,并以获得源程序基本模块为目标。

编码必须按照“详细设计说明书”的要求逐个模块地实现。在基于软件工程的软件开发过程中,编码往往只是一项语言转译工作,即把详细设计中的算法描述语言转译成某种适当的高级程序设计语言或汇编语言。

为了方便程序调试,针对基本模块的单元测试也往往和编码结合在一起进行。单元测试也以“详细设计说明书”为依据,用于检验每个基本模块在功能、算法与数据结构上是否符合设计要求。

### 4. 系统集成测试

所谓系统集成也就是根据概要设计中的软件结构,把经过测试的模块,按照某种选定的集



成策略，例如渐增集成策略，将系统组装起来。

在组装过程中，需要对整个系统进行集成测试，以确保系统在技术上符合设计要求，在应用上满足需求规格要求。

### 5. 系统确认验证

在完成对系统的集成之后，接着还要对系统进行确认验证。

系统确认验证需要以用户为主体，以需求规格说明书中对软件的定义为依据，由此对软件的各项规格进行逐项地确认，以确保已经完成的软件系统与需求规格的一致性。为了方便用户在系统确认期间能够积极参入，也为了系统在以后的运行过程中能够被用户正确使用，这个时期往往还需要以一定的方式对用户进行必要的培训。

在完成对软件的验收之后，软件系统可以交付用户使用，并需要以“项目开发总结报告”的书面形式对项目进行总结。

## 2.1.3 软件运行与维护期

软件系统的运行是一个比较长久的过程，跟软件开发机构有关的主要任务是对系统进行经常性的有效维护。

软件的维护过程，也就是修正软件错误，完善软件功能，由此使软件不断进化升级的过程，以使系统更加持久地满足用户的需要。因此，对软件的维护也可以看成为对软件的再一次开发。在这个时期，对软件的维护主要涉及三个方面的任务，即改正性维护、适应性维护和完善性维护。

## 2.2 瀑布模型

瀑布模型诞生于 20 世纪 70 年代，是最经典的并获得最广泛应用的软件过程模型。图 2-1 是传统瀑布模型的图样表示。

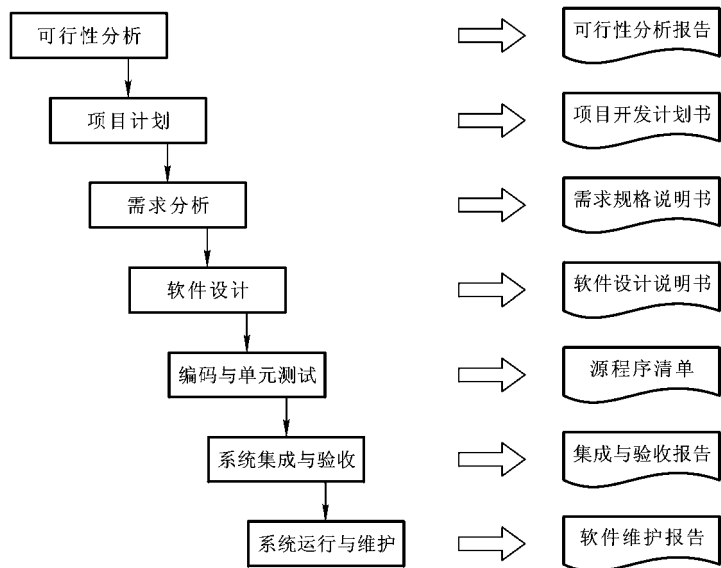


图 2-1 瀑布模型

瀑布模型中的“瀑布”是对这个模型的形象表达，即山顶倾泻下来的水，自顶向下、逐层细化。其中，自顶向下中的顶，可以理解为软件项目初期对软件问题的模糊认识，需要经过需求分析，才能使软件问题逐步清晰，而获得对软件规格的明确定义，由此使软件项目由定义期过渡到开发期，并经过软件开发而最终得到需要实现的软件产品这个最底层结果。瀑布模型中的逐层细化，其含义则是对软件问题的不断分解而使问题不断具体化、细节化，以方便问题的解决。

### 2.2.1 瀑布模型的特点

#### 1. 线性化模型结构

瀑布模型所考虑的软件项目是一种稳定的线性过程。项目被划分为从上至下按顺序进行的几个阶段，阶段之间有固定的衔接次序，并且前一阶段输出的成果被作为后一阶段的输入条件。

#### 2. 各阶段具有里程碑特征

瀑布模型中的阶段只能逐级到达，不能跨越。每个阶段都有明确的任务，都需要产生出确定的成果。

#### 3. 基于文档的驱动

文档在瀑布模型中是每个阶段的成果体现，因此，文档也就成为了各个阶段的里程碑标志。由于后一阶段工作的开展是建立在前一阶段所产生的文档基础之上，因此，文档也就成为了推动下一阶段工作开展的前提动力。

#### 4. 严格的阶段评审机制

在某个阶段的工作任务已经完成，并准备进入到下一个阶段之前，需要针对这个阶段的文档进行严格的评审，直到确认以后才能启动下一阶段的工作。

### 2.2.2 瀑布模型的作用

瀑布模型是一种基于里程碑的阶段过程模型，它所提供的里程碑式的工作流程，为软件项目按规程管理提供了便利，例如，按阶段制定项目计划，分阶段进行成本核算，进行阶段性评审等；并对提高软件产品质量提供了有效保证。

瀑布模型的作用还体现在文档上。每个阶段都必须完成规定的文档，并在每个阶段结束前都要对所完成的文档进行评审。这种工作方式有利于软件错误的尽早发现和尽早解决，并为软件系统今后的维护带来了很大的便利。

应该说，瀑布模型作为经典的软件过程模型，为其他过程模型的推出提供了一个良好的拓展平台。

### 2.2.3 带有信息反馈环的瀑布模型

在实际的软件项目中存在着许多不稳定因素。例如，开发中的工作疏漏或通信误解；在项目实施中途，用户可能会提出一些新的要求；开发者也可能在设计中遇到某些未曾预料的实际困难，希望在需求中有所权衡等。

考虑到许多实际项目中阶段之间有通信的需要，也就有了一种经过改进的，跟实际开发环境更加接近的瀑布模型，如图 2-2 所示。改进后的瀑布模型带有信息反馈环，能够逐级地将后

续阶段的意见返回，并在问题解决之后，再逐级地将修正结果下传。

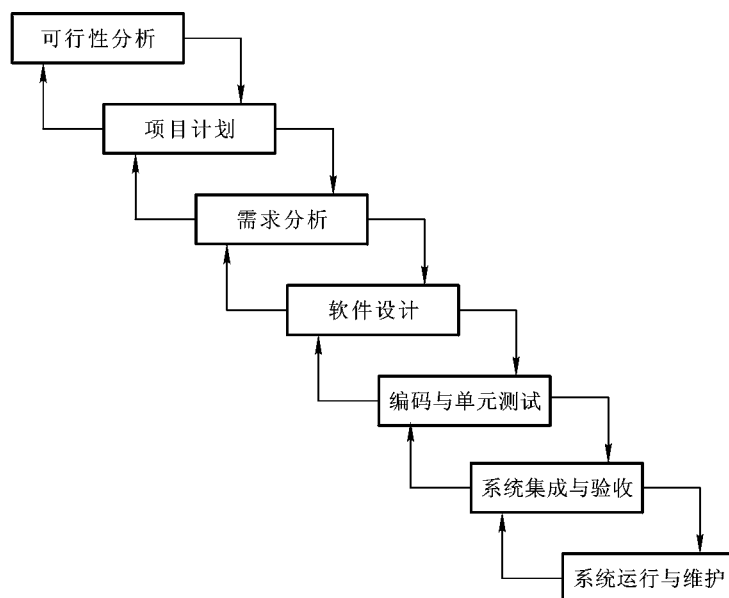


图 2-2 带有信息反馈环的瀑布模型

需要注意的是，为了确保文档内容的一致性，信息反馈过程中任何有关影响文档变更的行为，只能在相邻阶段之间逐级地进行。

## 2.2.4 瀑布模型的局限

瀑布模型是一种线性模型，要求项目严格按规定推进，必须等到所有开发工作全部作完以后才能获得可以交付的软件产品。应该讲，通过瀑布模型并不能对软件系统进行快速创建，对于一些急于交付的软件系统的开发，瀑布模型有操作上的不便。

瀑布模型主要适合于需求明确，且无大的需求变更的软件开发，例如，编译系统、操作系统等。但是，对于那些分析初期需求模糊的项目，例如那些需要用户共同参加需求定义的项目，瀑布模型也有使用上的不便。

## 2.3 原型模型

### 2.3.1 快速原型方法

快速原型方法是原型模型在软件分析、设计阶段的应用，用来解决用户对软件系统在需求上的模糊认识，或用来试探某种设计是否能够获得预期结果。

快速原型方法具有以下一些特点：

(1) 快速原型是用来获取用户需求的，或是用来试探设计是否有效的。一旦需求或设计确定下来了，原型就将被抛弃。因此，快速原型要求快速构建、容易修改，以节约原型创建成本、加快开发速度。快速原型往往采用一些快速生成工具创建，例如 4GL 语言。目前，Microsoft

Visual Basic、Inprise Delphi 等基于组件的可视化开发工具，也被应用于原型创建之中，并且都是非常有效的快速原型创建工具，而且还可用于原型进化。

(2) 快速原型是暂时使用的，因此并不要求完整。它往往针对某个局部问题建立专门原型，如界面原型、 workflow 原型、查询原型等。

(3) 快速原型不能贯穿软件的整个生命周期，它需要和其他的过程模型相结合才能产生作用。例如，在瀑布模型中应用快速原型，以解决瀑布模型在需求分析时期存在的不足。

### 2.3.2 原型进化模型

原型进化对开发过程的考虑是，针对有待开发的软件系统，先开发一个原型系统给用户使用，然后根据用户使用情况的意见反馈，对原型系统不断修改，使它逐步接近并最终到达开发目标。跟快速原型不同的是，快速原型在完成需求定义后将被抛弃，而原型进化所要创建的原型则是一个今后将要投入应用的系统，只是所创建的原型系统在功能、性能等方面还有许多不足，还没有达到最终开发目标，需要不断改进。

原型进化的工作流程如图 2-3 所示。从图中可以看到，它具有以下两个特点：

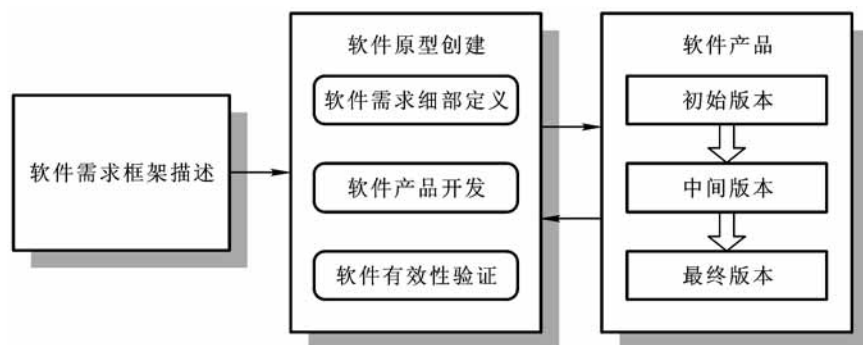


图 2-3 原型进化模型

(1) 原型进化模型将软件的需求细部定义、产品开发和有效性验证放在同一个工作进程中交替或并行运作。因此，在获得了软件需求框架以后，例如软件的基本功能被确定以后，就可以直接进入对软件的开发中。

(2) 原型进化模型是通过不断发布新的软件版本而使软件逐步完善的，因此，这种开发模式特别适合于那些用户急需的软件产品开发。它能够快速地向用户交付可以投入实际运行的软件成果，并能够很好地适应软件用户对需求规格的变更。

原型进化模型能够适应软件需求的中途变更，但在应用的时候，以下问题需要得到足够的重视。

其一，原型进化模型虽说使开发进程加快了，但不能像瀑布模型那样提供明确的里程碑管理，随着开发过程中版本的快速更新，项目管理、软件配置管理会变得复杂起来，管理者难以把握开发进度。因此，对于大型软件项目，原型进化模型缺乏有效的管理规程。

其二，开发过程中软件版本的快速变更，还可能损伤软件的内部结构，使其缺乏整体性和稳定性。另外，用于反映软件版本变更的文档也有可能跟不上软件的变更速度。这些问题必将

影响到今后软件的维护。

## 2.4 增量模型

瀑布模型较难适应用户的需求变更，开发速度慢。但是，瀑布模型提供了一套工程化的里程碑管理模式，能够有效保证软件质量，并使得软件容易维护。

相反地，原型进化模型则可以使对软件需求的详细定义延迟到软件实现时进行，并能够使软件开发进程加速。但是，原型进化模型不便于工业化流程管理，也不利于软件结构的优化，并可能使得软件难以理解和维护。

基于以上因素的考虑，增量模型对这两种模型的优点进行了结合。

### 2.4.1 增量模型的特点

增量模型是瀑布模型和原型进化模型的综合，它对软件过程的考虑是：在整体上按照瀑布模型的流程实施项目开发，以方便对项目的管理；但在软件的实际创建中，则将软件系统按功能分解为许多增量构件，并以构件为单位逐个地创建与交付，直到全部增量构件创建完毕，并都被集成到系统之中交付用户使用。

如同原型进化模型一样，增量模型逐步地向用户交付软件产品，但不同于原型进化模型的是，增量模型在开发过程中所交付的不是完整的新版软件，而只是新增加的构件。

图 2-4 是增量模型的工作流程，它被分成以下三个阶段：

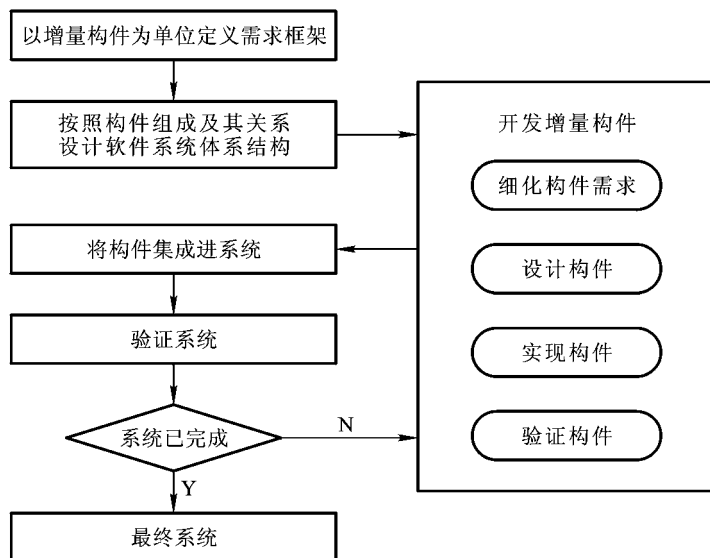


图 2-4 增量模型

(1) 在系统开发的前期阶段，为了确保所建系统具有优良的结构，仍需要针对整个系统进行需求分析和概要设计，需要确定系统的基于增量构件的需求框架，并以需求框架中构件的组

成及关系为依据，完成对软件系统的体系结构设计。

(2) 在完成软件体系结构设计之后，可以进行增量构件的开发。这个时候，需要对构件进行需求细化，然后进行设计、编码测试和有效性验证。

(3) 在完成了对某个增量构件的开发之后，需要将该构件集成到系统中去，并对已经发生了改变的系統重新进行有效性验证，然后再继续下一个增量构件的开发。

## 2.4.2 增量模型的作用

增量模型带来了以下几方面的作用。

(1) 开发初期的需求定义只是用来确定软件的基本结构，这使得开发初期，用户只需要对软件需求进行大概的描述，而对于需求的细节性描述，则可以延迟到增量构件开发时进行，以增量构件为单位逐个地进行需求补充。这种方式有利于用户需求的逐渐明朗，能够有效适应用户需求的变更。

(2) 软件系统可以按照增量构件的功能安排开发的优先顺序，并逐个实现和交付使用。这不仅有利于用户尽早地用上系统，能够更好地适应新的软件环境，而且用户在以增量方式使用系统的过程中，还能够获得对软件系统后续构件的需求经验。这样能使软件需求定义越往后越顺利。

(3) 软件系统是逐渐扩展的，因此，开发者可以通过对诸多构件的开发，逐步积累开发经验。实际上，增量式开发还有利于技术复用，前面构件中设计的算法、采用的技术策略、编写的源码等，都可以应用到后面将要创建的增量构件中去。

(4) 增量式开发还有利于从总体上降低软件项目的技术风险。个别的构件或许不能使用，但这一般不会影响到整个系统的正常工作。

(5) 实际上，在采用增量模型时，具有最高优先权的核心增量构件将会被最先交付，而随着后续构件不断被集成进系统，这个核心构件将会受到最多次数的测试。这意味着软件系统最重要的核心部分将具有最高的可靠性，这将使得整个软件系统更具健壮性。

可以说，比较瀑布模型、原型进化模型，增量模型具有非常显著的优越性。但是，增量模型对软件设计有更高的技术要求，特别是对软件体系结构，要求它具有很好的开放性与稳定性，能够顺利地实现构件的集成。在把每个新的构件集成到已建软件系统的结构中的时候，一般要求这个新增的构件应该尽量少地改变原来已建的软件结构。因此增量构件要求具有相当好的功能独立性，其接口应该简单，以方便集成时与系统的连接。

## 2.5 螺旋模型

软件开发过程中存在许多方面的风险。例如，软件设计时遇到了很难克服的技术难题，软件开发成本超出了先期预算，软件产品不能按期交付，用户对所交付的软件不满意等。应该说，软件风险是任何软件项目中都普遍存在的实际问题，而且项目越大，软件越复杂，风险也就越大。由于软件风险可能在不同程度上损害软件开发过程，并由此影响软件产品质量，因此，在软件开发过程中需要及时地识别风险、有效地分析风险，并能够采取适当措施消除或减少风险的危害。



## 2.6 喷泉模型

喷泉模型是专门针对面向对象软件开发方法而提出的。“喷泉”一词用于形象地表达面向对象软件开发过程中的迭代和无缝过渡。

在面向对象方法中，对象既是对现实问题中实体的抽象，也是构造软件系统的基本元素。因此，建立对象模型在面向对象方法中，既可以用于分析，也可以用于设计，而且分析阶段所获得的对象框架模型可以无缝过渡到设计阶段，以作为软件实现的依据。

喷泉模型的过程方法所考虑的是，基于面向对象方法所带来的便利，对软件的分析、设计和实现按照迭代的方式交替进行，并通过进化的方式，使软件分阶段逐渐完整、逐步求精。例如，第一阶段软件开发的目标可以是软件的基本功能；第二阶段可以是在第一阶段建立的软件的基础上，对软件进行进一步的完善，并实现软件的主要功能；第三阶段则是在第二阶段的基础上，对软件进行更加完整的开发，并以实现软件全部功能作为创建目标。

应该说，喷泉模型能够较有效地平衡软件系统的近期需求与远期规划，因此能够较好地满足用户在软件应用上的发展需要。

## 2.7 组件复用模型

自有软件开发以来，软件复用就一直存在，并产生了一些比较常用的软件复用方法。传统的软件复用方法是建立源程序函数库，可以把这些函数用在许多不同的软件上面。而在面向对象技术中，软件复用则可以通过建立类模块来实现。由于大多数的类模块具有继承性，因此，比起传统方法，基于类模块的复用效果要好一些。

组件复用方法是最近几年才发展起来的更加先进的软件复用技术，它能带来更好的复用效果，并且更具有工程特性，更能适应软件按工业流程生产的需求。

组件技术是基于面向对象技术发展起来的，可以把组件看作为一个盒子，它里面封装了许多个类模块。因此，组件比类更大、更抽象，其中包含了更多的功能，更具有通用性，更加有利于复用。

在基于组件复用的软件开发中，软件由组件装配而成，这就如同用标准零件装配汽车一样。图 2-6 是组件复用模型的工作流程，它以组件复用为驱动。

组件复用模型主要包含以下几个阶段的工作任务。

(1) 需求框架描述：描述软件系统功能构成，并将各项功能以设定的组件为单位进行区域划分。

(2) 组件复用分析：按照需求框架中的组件成分，分析哪些组件是现成的，哪些组件可以在专业组件开发机构购买到，哪些组件不得不自己开发。

(3) 需求修改与细化：以提高对现有组件的复用和降低新组件开发为目标，调整需求框架，并对已经确定的需求框架进行细化，

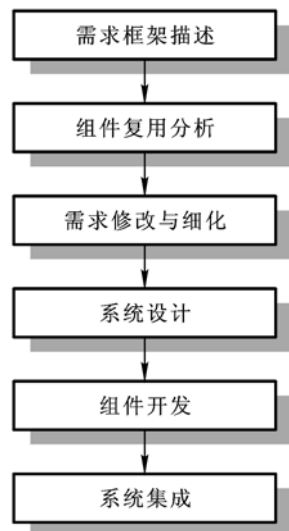


图 2-6 组件复用模型



由此获得对软件系统的详细需求定义。

(4) 系统设计：基于组件技术设计系统框架，设计需要开发的新组件。

(5) 组件开发：开发不能获得复用的新组件。

(6) 系统集成：根据设计说明要求，将系统所需要的诸多组件整合在一起，构成一个完整的系统。

应该说，组件复用技术给软件开发带来了许多好处。它可以缩短开发周期、降低开发成本、提高软件质量、方便软件维护，特别是可以使软件生产按工业流程进行。

但值得注意的是，在组件复用中有可能会遇到组件复用率与用户特殊需求之间的矛盾。例如，开发机构希望使用现有的组件开发软件以降低开发成本，但现有组件却可能不能满足用户的特殊需求。

另外，当采用组件复用技术实施软件开发时，软件中需要用到的组件往往是由一些专门的软件机构提供。也就是说，基于组件的软件开发对外有很大的依赖性，显然这不太利于软件的升级、改版。

## 小 结

### 1. 软件生命周期

如同任何事物都有一个发生、发展、成熟直至衰亡的全过程一样，软件系统或软件产品也有一个定义、开发、运行维护直至被淘汰这样的全过程，我们把软件将要经历的这个全过程称为软件的生命周期。它包含：软件定义、软件开发、软件运行维护三个时期，并可以细分为可行性研究、项目计划、需求分析、概要设计、详细设计、编码实现与单元测试、系统集成测试、系统确认验证、系统运行与维护等几个阶段。

### 2. 瀑布模型

瀑布模型诞生于 20 世纪 70 年代，是最经典的并获得最广泛应用的软件过程模型。瀑布模型中的“瀑布”是对这个模型的形象表达，即山顶倾泻下来的水，自顶向下、逐层细化。

(1) 特点：线性化模型、阶段具有里程碑特征、基于文档的驱动、阶段评审机制。

(2) 作用：为软件项目按规范管理提供了便利，为其他过程模型的推出提供了一个良好的拓展平台。

(3) 局限性：主要适合于需求明确且无大的需求变更的软件开发，但不适合分析初期需求模糊的项目。

### 3. 原型模型

(1) 快速原型方法：是原型模型在软件分析、设计阶段的应用，用来解决用户对软件系统在需求上的模糊认识，或用来试探某种设计是否能够获得预期结果。

(2) 原型进化模型：针对有待开发的软件系统，先开发一个原型给用户使用，然后根据用户的使用意见，对原型不断修改，使它逐步接近，并最终到达开发目标。

### 4. 增量模型

增量模型结合了瀑布模型与原型进化模型的优点。在整体上按照瀑布模型的流程实施开发，以方便对项目的管理。但在软件的实际创建中，则将软件系统按功能分解为许多增量构件

逐个地创建与交付，直到全部构件创建完毕，并都被集成到系统之中交付使用。

比较瀑布模型、原型进化模型，增量模型具有非常显著的优越性。但增量模型对软件设计有更高的技术要求。

#### 5. 螺旋模型

螺旋模型是一种引入了风险分析与规避机制的过程模型，是瀑布模型、快速原型方法和风险分析方法的有机结合。其基本方法是，在各个阶段创建原型进行项目试验，以降低各个阶段可能遇到的项目风险。

#### 6. 喷泉模型

喷泉模型是专门针对面向对象软件开发方法而提出的。“喷泉”一词用于形象地表达面向对象软件开发过程中的迭代和无缝过渡。

#### 7. 组件复用模型

组件复用方法是最近几年发展起来的先进的软件复用技术，在基于组件复用的软件开发中，软件由组件装配而成，这就如同用标准零件装配汽车一样。因此，组件复用模型能够有效地提高软件生产率。

## 习 题

1. 什么是软件生命周期？根据国家标准《计算机软件开发规范》，软件生命周期主要包括哪些阶段？
2. 瀑布模型有哪些特点？对于里程碑，你有什么认识？一般认为，瀑布模型不太适用于用户需求经常变更的软件项目，其原因是什么？
3. 试说明快速原型的作用。
4. 原型进化模型是一种与瀑布模型有着显著差别的软件过程模型。与瀑布模型相比，其优点是什么？一般认为，原型进化模型不能适应较大型软件项目的开发，其原因是什么？
5. 增量模型是一种结合了瀑布模型与原型进化模型共同优点的过程模型，其特点是什么？在使用增量模型进行软件开发时需要注意的问题是什么？
6. 试说明螺旋模型的特点。一般认为，只有大型项目才有采用螺旋模型的必要，其原因是什么？
7. 喷泉模型是专门针对面向对象软件开发方法而提出的，其特点是什么？
8. 为什么说组件复用模型是一种有利于软件按工业流程生产的过程模型？
9. 某大型企业计划开发一个“综合信息管理系统”，涉及销售、供应、财务、生产、人力资源等多个部门的信息管理。该企业的想法是按部门优先级别逐个实现，边应用边开发。对此，需要一种比较合适的过程模型。请对这个过程模型作出符合应用需要的选择，并说明选择理由。

# 第 3 章 项目分析与规划

在软件项目早期，需要对软件问题进行高层构架分析，以确定项目的可行性。还需要根据可行性分析的结果制定出有效的项目实施计划，以指导软件项目的顺利开展。本章将要讨论的是这些项目的前期准备性工作。

## 3.1 计算机系统设计

当某个软件问题被作为项目提出时，即意味着，这个软件问题将成为一项工程任务，需要按照工程化作业流程来分阶段解决。其中，计算机系统设计是软件项目工程化作业流程中首先需要进行的预备性工作，它以整个计算机系统作为分析背景，由此得到对有待开发的软件系统及其工作环境的全貌性的了解。

### 3.1.1 计算机系统

系统是一个群体概念，一个事物若被称为系统，则表明这个事物之内包含有许多元素。系统由元素组成，但系统并不是其所包含的元素的简单组合，而是一个有机整体，其内部元素之间相互关联、协同工作，具有一致的任务目标。

有些系统是简单的，比如钢笔，它仅由笔尖、吸墨器、笔杆、笔套等简单零件组成，其构造可以一目了然。然而有些系统则是复杂的，比如汽车，具有复杂的内部构造。

一般来说，比较复杂的系统大都具有分层构造，其内部元素也具有系统特征，可以作为子系统看待。计算机系统就是一个非常复杂的系统，包含有：硬件系统、软件系统、网络通信系统、人工操作系统等诸多子系统，而其中的软件系统又由操作系统、数据库系统、应用程序等更小的系统元素组成，如图 3-1 所示。

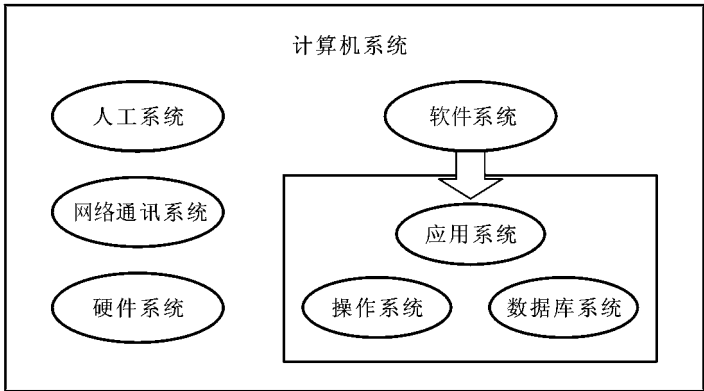


图 3-1 计算机系统构架

计算机系统还是一个具有智能特性的开放系统。计算机的智能特性就是通过它的软件系统提供的，一个新的软件的诞生，实质上也就是计算机又得到了一次智能扩充。

### 3.1.2 系统分析方法

软件项目是基于计算机的系统工程，因此，针对软件项目的系统分析需要以计算机这个更大的系统为背景进行，需要将软件问题放置到整个计算机系统去看。因此，针对有待开发的软件系统的前期分析，也就不只是软件系统本身了，它还需要分析与有待开发的软件系统有关的其他方面的问题，例如，硬件环境、网络环境、支撑软件等。

系统分析是对项目的高层分析，需要得到的是系统的基本框架。因此，系统分析并不需要对有待开发的软件系统的内部构造作出过多的研究，但需要对系统按能够独立工作的组件或子系统为单位进行分解，由此使系统能够从它所处的环境中分离出来，其结果将作为划分系统边界与确定系统框架结构的基本依据。

系统分析也不需要研究有待开发的软件系统的控制机制，但却有必要以独立组件或子系统为单元，研究系统的工作流程、系统与外界的数据通信。

具体说来，系统分析可以从以下几个方面对软件及其相关问题作出描述：

1. 软件系统的规模大小、功能范围。
2. 软件系统对硬件环境、网络环境、数据环境和支撑软件的依赖。
3. 软件系统中有关安全保密问题的考虑。
4. 软件系统与其他相关系统之间的数据通信。
5. 软件用户，包括：用户单位的组织结构，与软件有关的用户的工作流程。

### 3.1.3 建立系统模型

建立系统模型，也就是以作图方式对系统进行直观的描述。计算机系统前期分析过程中用得比较多的图形模型有：系统框架图、系统流程图。其中，系统框架图用于描述系统的基本框架，系统流程图用于描述系统的基本工作流程。下面分别加以介绍。

#### 1. 系统框架图

系统框架图使用矩形与带箭头的线段来描述系统的基本体系构造，能够用来描述系统的逻辑框架。

系统框架图中的矩形用于表示构造系统的组件或子系统，线段则用于表示组件或子系统之间的关系。例如图 3-2 所示的自动阅卷系统的系统框架图，它所表达的是：自动阅卷系统由成绩数据库、读卡程序、阅卷程序、成绩查询程序、成绩分析程序、成绩单打印程序和成绩备份程序组成，并以成绩数据库为核心构建。

一般情况下，可以使用系统框架图说明系统的功能分配和划分系统边界。

#### 2. 系统流程图

系统流程图被用来描述系统的工作流程，以系统中的物理组件为单元说明系统的基本构造，并由此说明系统对数据的加工步骤。表 3-1 所列是系统流程图中常用的图形符号。显然，系统流程图中的符号是一些可以从系统中分离出来的物理元素，例如，设备、程序模块、报表等。

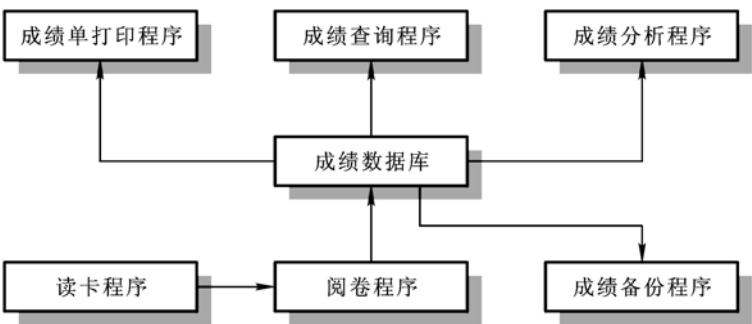


图 3-2 自动阅卷系统框架图

表 3-1 常用的系统流程图的图形符号

图 形 符 号	说 明	图 形 符 号	说 明
	任何处理，包括程序处理、机器处理、人工处理等		顺序存储，例如磁带存储
	人工处理，例如会计在银行支票上的签名		纸带存储
	任何输入，包括人工输入、程序输入等		文档输出，例如打印数据报表
	卡片输入		显示终端
	手工输入		页内连接
	任何存储		换页连接
	内部存储		数据流
	直接存储，例如磁盘存储		

图 3-3 是关于“自动阅卷系统”的系统流程图，它所表达的是：阅卷系统首先通过读卡程序读入考试答题卡片上的数据，然后经过阅卷程序计算出成绩并存入成绩数据库，在产生出成绩数据之后，系统可以通过成绩查询程序、成绩分析程序、成绩单打印程序和成绩备份程序，对成绩数据库中的数据进行相关操作。

显然，与系统框架图比较，系统流程图包含了更多的数据加工细节，并能通过物理元素表现系统的物理构架，以及与外部接口的连接通信等。

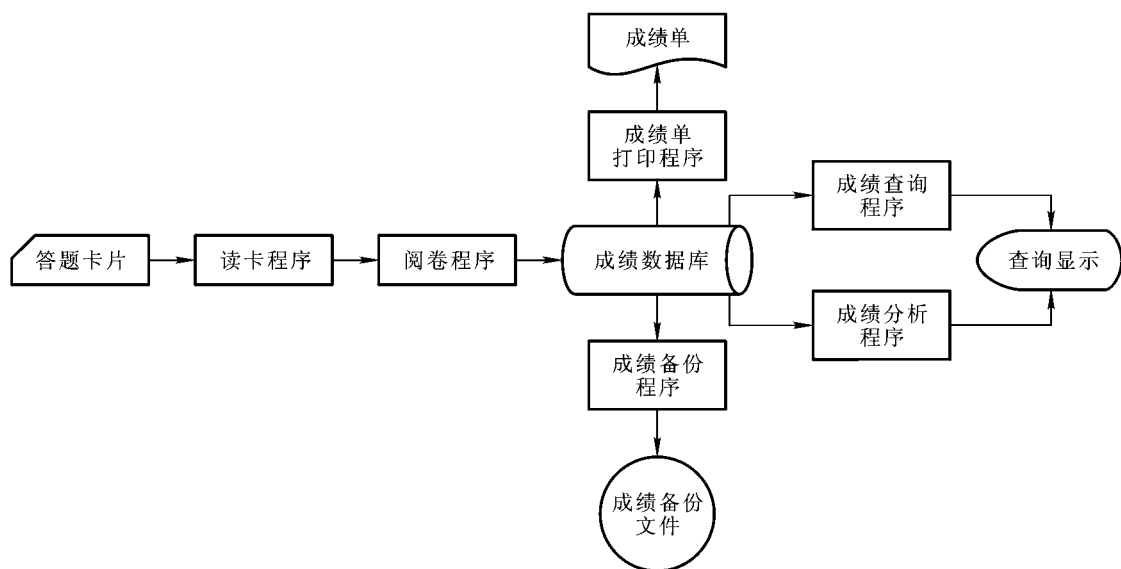


图 3-3 “自动阅卷系统”的系统流程图

## 3.2 项目可行性分析

### 3.2.1 可行性分析意义

软件问题往往以任务立项的形式被作为工程提出，这是软件项目实施过程中需要进行的又一项准备工作，涉及对软件问题的一些概括性描述。例如，项目的名称、性质和意义，项目的任务来源，有待开发的软件在功能、性能等方面需要达到的基本目标，项目在资金、设备和技术支持等方面所受到的条件限制等。

软件任务立项可以被看作为具有宣言意义的纲领，以最简洁的方式，从项目的最高层面上表达了项目需要达到的目标。

但需要注意的是，凡是工程项目就必然具有工程风险，其中软件项目则更是具有高风险的特征。这也意味着，软件项目很有可能不能够达到任务立项时所预期的工程目标。因此，在软件项目正式实施之前，有必要对其进行可行性分析，由此对软件项目是否能够达到任务立项所预期的工程目标作出一个比较明确的是非判断。

工程经验表明：可行性分析对软件问题解决途径的探索，能够给软件项目带来以下方面具有积极意义的影响。

（1）通过少量的费用，对项目能否实施尽早作出决断，以避免项目开展以后所带来的大量的人力、物力和时间的浪费。

（2）根据项目所受到的条件限制，对有待开发的系统在体系构造、工作模式等方面作出高层抉择，以利于项目今后的实现。

（3）可以把可行性分析看作软件定义时期需要进行的前导性工作，其结果可以作为一个高层框架被用于软件需求分析过程之中，以方便今后软件规格定义工作的顺利开展。

### 3.2.2 可行性分析内容

软件开发总是离不开对软件中技术因素的考虑。但是，当软件问题成为工程问题时，它还将受到来自项目费用预算、开发期限和应用环境等其他方面因素的制约。因此，对软件项目的可行性分析，也就包括了技术、经济、应用等诸多方面的分析。

#### 1. 技术可行性

技术可行性是关于软件项目技术问题的高层策略，涉及对有待开发系统的高层技术构架的探索。技术可行性需要确认的是：项目所准备采用的技术是先进的、成熟的，能够充分满足用户系统在教育上的需要，并足以从技术上支持系统的成功实现。

对技术可行性的判断是基于软件开发者可能采用的技术而提出的，并需要从技术与技术资源这两个方面作出可行性评估。

(1) 技术限制。一般来说，用于开发软件的技术会受到先进性、成熟度等因素的限制。例如，软件技术的先进性与成熟度是否能够保证开发出来的系统达到预期的功能、性能、安全等技术目标。其中，技术的先进性与成熟度是两个需要平衡考虑的技术因素。例如，开发者可能希望使用一种刚刚诞生不久的新技术，以使所开发的系统能够获得技术先进性所带来的开发上的便利，并使系统获得更好的应用效果。但是，刚诞生的新技术却有可能缺乏使其成熟的比较充分的工程验证，因此有可能会给项目带来比较高的技术风险。

(2) 技术资源限制。这是指开发机构对所采用的技术的把握程度方面的限制。例如，开发人员对技术的熟练程度，开发机构目前所拥有的软件资源、设备、工程经验和能够得到的技术支持等。显然，尽管将采用的技术既先进又成熟，但假如开发机构刚刚使用这种技术不久，其在该项技术的应用上工程经验还积累得不够，技术资料也相对比较缺乏，那么，从工程角度考虑，它仍有可能不具备可行性。

#### 2. 经济可行性

经济可行性分析是对项目实施成本和项目可能带来的经济效益的分析，以确定等待实施的项目是否值得投资。

应该说，任何项目都会受到项目资金的限制，软件项目也不例外。因此，针对软件项目必须进行成本估算，并以项目成本是否在项目资金限制范围以内作为项目的一项可行性依据。

影响经济可行性的另一个因素是项目的经济效益。实际上，无论是开发机构或是用户，都会关心项目效益，因此必须对项目进行效益评估，从中发现项目预期的经济收益是不是能够超过项目投资。

在分析项目效益时需要注意的是，开发机构与用户会有不同的效益来源，需要分别对待。其中，开发机构的效益直接来源于软件产品，而用户的效益则应该说是来自于对软件的间接应用。

#### 3. 应用可行性

应用方面的可行性将涉及法律法规、用户操作规程等方面的问题。法律法规方面的问题有：开发的软件会不会构成法律侵权，会不会跟国家的相关政策、法律发生冲突，开发者、用户双方的职责和权利有没有通过合同等法律形式明确体现出来等。用户操作规程方面的问题有：用户单位的组织结构、与软件有关的用户工作流程、用户在计算机应用上的素质等。

软件是开发给用户使用的，因此应该考虑所开发的软件是否能够跟用户单位已经形成的管理模式、工作流程、操作方式等保持良好的协调。例如，基于传统的 C/S 结构的局域网上的信息系统，由于具有较好的工作性能和便利的操作方式，而且技术成熟、容易实现，它比较适合一些地域集中的企业的信息化管理。但是，一个跨区域的企业，由于需要通过互联网进行通信，C/S 结构就可能不太合适了，更加合适的结构可能是基于 B/S 结构的 Web 信息系统。

### 3.2.3 可行性分析过程

可行性分析的一般步骤是：建立系统模型，进行项目可行性评估，撰写可行性分析报告。下面将介绍这几个步骤。

#### 1. 建立系统模型

可行性分析所针对的是一个尚未创建的新系统，为了使可行性分析具有研究对象，在进行可行性评估之前有必要先建立起该系统的工作模型。

建立系统模型可以按照以下步骤进行。

(1) 研究现有系统的物理模型。有足够的理由使人相信，新的系统将建立在现有系统基础之上。例如，新系统与现有系统由相同的用户使用，新系统与现有系统涉及了一些相同的工作内容，现有系统积累下来的数据，将要由新的系统继续给予维护等。因此，在建立新系统模型时，有必要研究现有系统的工作模型。

需要注意的是：现有系统是一个处在工作状态的现实系统，因此，对现有系统的研究，可以从它的物理模型着手，例如，可以使用系统流程图来对它做直观的物理描述。

(2) 导出现有系统的逻辑模型。对现有系统的研究是为了更加合理地对新系统提出可供参考的实施方案，但是，物理模型是一种相对固化的模型，当需要由现有系统对新系统进行推断时，物理模型不如逻辑模型灵活。因此，为了方便地构造新系统的工作模型，有必要将现有系统的物理模型转化成为它的逻辑模型，例如，从现有系统的系统流程图模型推导出它的数据流程图模型。

(3) 设想新系统的逻辑模型。逻辑模型具有变通性，因此，可以将现有系统的逻辑模型作为前提条件，并以新系统需要获得的功能为目标，对新系统的逻辑模型作出合理的设想。

(4) 提出新系统的物理模型。可行性研究是对项目能否实施、软件能否实现的可行性评估。由于软件系统的实现最终都将体现为某种具体的物理形态，因此，为了使得可行性研究所提出的建设性意见更加具有可信度，还需要以系统的逻辑模型为依据，提出有关系统的物理模型。

需要注意的是：由于逻辑模型所体现出来的只是系统的功能构架，往往可以采用许多种不同的物理方案给以具体实现。因此，基于同一个逻辑模型，有必要提出几个适当的物理模型供项目参考。

#### 2. 进行可行性评估

随着系统初步模型的产生，可行性分析已经有了一个适当的研究对象。

在新系统的物理方案被提出来以后，接下来需要进行的工作就是针对所提出的物理方案，从技术、经济、应用等方面进行可行性评估，并由此归纳出对于软件项目的总的可行性评估结论。

下面是一些可供参考的可行性评估结论：

(1) 项目各方面条件都已经基本具备，并能够产生很好的影响或获取有效的成果，建议立



即着手实施项目。

(2) 主要条件已基本具备,但准备工作仍显得不充分,例如,项目管理中还缺少专门的质量监控机构,有关技术资源不够充分等,建议在相关条件得到进一步完善之后,再着手实施项目。

(3) 项目实施的基本条件不具备,例如,项目资金缺口太大、项目技术难以在限定时间内有所突破等,建议中止项目。

### 3. 撰写可行性分析报告

可行性分析是软件工程过程中的一个重要阶段,因此可行性分析中产生的一系列结论要以文本形式体现出来。例如,按照一定的格式撰写成“可行性分析报告”,并报送有关部门或机构(例如:用户单位、项目领导小组等)进行审核。

## 3.3 项目成本效益分析

经济可行性研究是对项目实施成本和所能带来的经济效益的分析,以确定等待实施的项目是否值得投资。

### 3.3.1 项目成本估算

在项目初期,无论是进行可行性分析,还是制定项目预算,或是向客户提供软件报价,都需要针对软件项目进行成本的初步估算。下面将要介绍的是一些常用的软件项目成本估算方法。

#### 1. 基于软件规模的成本估算

传统的软件规模是通过代码行数计算的。也就是说,通过估算软件代码总行数,可以计算出创建软件的总工作量和软件总成本。

基于软件代码行数的人力成本估算公式是:

$$WC = (TCL / MPACL) \times MPAP$$

计算公式中的  $WC$  是软件工作成本,  $TCL$  是软件总代码行数,  $MPACL$  是以参入项目所有人员为基数计算的每月人均完成的代码行数,  $MPAP$  是以参入项目所有人员为基数计算的每月人均工资。其中,参入项目所有人员既包括技术人员,也包括管理人员。

在对软件代码行数进行估算时,往往需要先将软件按功能进行分解。例如,可以将软件系统按照功能分解为许多子系统,子系统又可以继续分解为许多功能模块,这种对软件系统的分解工作可以一直进行到基本模块。应该说,基本模块的代码行数是比较容易估算的,而通过对基本模块代码行数进行估算与累计,可以估算出整个系统的总代码行数。

代码行数估算方法比较适合一些需求比较容易确定,并且采用传统开发工具(例如:汇编语言、C 语言等)开发的系统。但在目前,更多的应用项目是面向用户的,项目非常庞大,软件需求在项目初期也往往只有一个框架。开发这些系统所采用的开发工具则一般是面向对象或基于组件的,例如: C++、Java、Visual Basic 等;并且在软件实现过程中,还有可能会大量地应用到诸多重用技术,例如,将一些已经存在的对象或组件引用到当前新的项目之中来。这些都给代码行数估算方法带来了困难。对于这样的软件系统,一种更加合适的软件规模估算是基于面向用户的应用对象进行的,例如,用户的操作窗口、提供打印的数据报表、用于数据计算的功能组件、数据库中的数据表或数据视图等。这个时候,诸多复杂程度不同的应用对象可

以采用合适的对象点数加以标识，例如，简单的用户窗口其对象点数取“1”；中等复杂程度的用户窗口其对象点数取“2”；而非常复杂的用户窗口其对象点数则可以取“3”等。

显然，比起传统的代码行数估算方法来，基于应用对象的软件规模估算，能够更加方便地在项目初期对那些只是进行了高层结构描述的软件系统进行成本估价。

基于应用对象的人力成本估算公式是：

$$WC = ( ( NOP ( 1 - R ) ) / PROP ) \times MPAP$$

计算公式中的  $WC$  是软件工作成本， $NOP$  是应用对象的对象点数， $R$  是构造应用对象的代码复用率， $PROP$  是以参入项目所有人员为基数计算的每月人均完成的对象点数， $MPAP$  是以参入项目所有人员为基数计算的每月人均工资。其中，参入项目所有人员既包括技术人员，也包括管理人员。

2. 基于任务分解的成本估算

这是一种以项目任务的人力消耗为依据的成本估算方法。可以把项目任务分解成诸多活动，例如，按照工程过程将项目任务分解成需求分析、概要设计、详细设计等若干个阶段，然后根据每个阶段的人员配备、周期长短和阶段任务参加人员平均工资情况，而估算出每个阶段的人力成本，由此累计出项目总成本。

例如为某企业开发“企业资源综合管理系统”时考虑按表 3-2 所列进行人员配备。

表 3-2 开发“企业资源综合管理系统”时的人员配备情况

任 务 名 称	参入人员结构（人）	完成周期（月）
系统分析	系统分析师：1 结构设计师：1 文档管理员：1	1
体系结构设计	结构设计师：1 高级程序员：3 文档管理员：1	1
详细算法设计	高级程序员：3 基础程序员：6 文档管理员：1	1
编码	高级程序员：3 基础程序员：6 文档管理员：1	2
系统集成	结构设计师：1 高级程序员：3 文档管理员：1	2

表 3-3 所列为参入人员工资参照标准。

表 3-3 参入人员工资标准

人 员 类 别	月平均工资（元）	人 员 类 别	月平均工资（元）
系统分析师	6 000	基础程序员	1 500
结构设计师	5 000	文档管理员	2 000
高级程序员	3 000		

那么，根据上述数据可以计算出该项目的人力成本如表 3-4 所列。

表 3-4 开发“企业资源综合管理系统”时的人力成本情况

任 务 名 称	阶段成本（元）	任 务 名 称	阶段成本（元）
系统分析	13 000	编码	40 000
体系结构设计	17 000	系统集成	34 000
详细算法设计	20 000	人力成本总计	124 000

### 3. 成本估算中的其他因素

软件成本主要是工作成本，上述对软件成本的估算主要是基于软件开发的任务量计算的。尽管如此，软件项目中的其他因素也不能完全忽略不计，例如，计算机硬件设备费用，软件资源费用，差旅培训费用，开发场地、电力和网络通信费用等。在项目初期制定项目预算时，需要将诸多问题逐项列出，以防止由于计算遗漏而造成项目实施过程中出现经费短缺现象。

实际项目中，影响项目成本的因素可能更多，例如，软件的应用领域、开发环境、开发队伍、国家差别等。实际上，影响软件成本的诸多因素大都是一些模糊因素，这使得对项目成本的估算不可能很精确。为了避免成本估算中出现大的偏差，对成本的估算还可以考虑采用其他不同方法，并有必要运用统计技术。例如，邀请多位软件技术和应用领域方面的专家，由他们分别用其所熟悉的方法对项目做成本估算，然后再进行成本统计分析。

为了提高成本估算的准确性和便利性，开发机构有必要根据自己的项目经验或有关资料，建立项目成本模型，并依靠有关模型参数使诸多经验数据在成本估算中产生作用。其中，项目的类型、规模、难度、应用领域等，都可以作为量化参数。另外，在项目具体实施以后，还需要根据项目的实际进展情况和项目的需求变动等情况，对项目成本进行修正，这可以使对项目成本的估算更加接近实际。

#### 3.3.2 项目效益分析

无论是开发机构或是用户，都会关心项目效益，但值得注意的是，开发机构的效益直接来源于软件产品，而用户的效益则来自于对软件的应用。并且不同的软件产品会有不同的效益来源，例如，软件机构自主开发的通用软件 and 用户委托开发的定制软件，它们在效益来源上就分别有各自不同的途径。

通用软件由软件机构自主开发，然后投放到软件市场上销售。开发机构的最低期望可能是，软件在销售中所获取的直接经济利益至少能超过软件的开发成本，以保证收回投资。对于通用软件，开发机构大都需要在开发软件之前进行深入的市场分析，看软件市场是否已有了同类型的产品，假如已有同类型产品，则要看开发的产品在功能、性能、价格等方面是否具有市场优势等。

而对于由用户委托开发的软件项目，开发机构的效益则取决于用户对项目的资金投入与软件实际成本的差额值，其计算看起来是简单的。但是，这些项目由于完全由用户进行巨额投资，其效益也就必然受到用户的特别关注，所以计算起来非常复杂。用户的期望可能是，花费巨额资金开发出来的软件在其使用过程中能够提高工作效率、改善工作质量、节约工作成本、拓宽业务领域等，由此带来的间接经济效益至少能够超过软件的开发成本。

在计算项目的经济效益时，还不得不注意到，软件的经济效益是在软件投入使用之后的若干年里逐渐产生出来的，而资金投入则是当前之事。为了更加合理地计算资金效益，未来效益中产生的资金需要折算为现值进行计算。

资金折现公式是：

$$\text{资金折现值} = \text{资金未来值} / (1 + k)^n$$

其中， $k$ 是银行利率； $n$ 是年份。

可以使用一些经济指标来衡量项目的经济效益，其主要经济指标有：

(1) 纯收入：指软件在估算的正常使用期内产生的资金收益被折算为现值之后，再减去项目的成本投入。

(2) 投资回收期：指软件投入使用后产生的资金收益折算为现值，到项目资金收益等于项目的成本投入时所需要的时间。

(3) 投资回收率：指根据软件的资金收益进行利息折算，可以将其与银行利率做比较。

显然，若项目的投资回收期超过了所开发软件的正常使用期，或项目的投资回收率低于银行利率，或纯收入为负值，则项目在经济效益上不具有可行性。

例如某“企业资源综合管理系统”的开发。假设开发过程中，人力、设备、支撑软件等各项成本总计预算是20万，计划一年开发完成并投入使用。表3-5所列预计有效5年生命周期内的逐年经济收益与折现计算。其中，银行年利率按6%计算。

表3-5 “企业资源综合管理系统”逐年经济收益与折现计算

年	逐年收益(元)	$1/(1+0.06)^n$	折现值(元)
1	50 000	0.94	47 000
2	80 000	0.89	71 200
3	80 000	0.84	67 200
4	80 000	0.79	63 200
5	60 000	0.75	45 000
收益总计			293 600

由表3-5可以推算出以下结果：

(1) 纯收入 =  $293\,600 - 200\,000 = 93\,600$  (元)。

(2) 投资回收期 =  $3 + (200\,000 - 47\,000 - 71\,200 - 67\,200) / 63\,200 = 3.23$  (年)。

(3) 投资回收率的计算则相对比较复杂，需要通过一个高阶代数方程才能计算出来。当前软件问题的投资回收率计算的高阶代数方程是：

$$5/(1+j) + 8/(1+j)^2 + 8/(1+j)^3 + 8/(1+j)^4 + 6/(1+j)^5 = 20$$

### 3.4 项目规划

专业的软件开发总是要受到成本预算、工程进度、质量要求等因素的制约。因此，开发机构在项目实施之前需要对项目作出全面的规划，并通过计划书的形式明确体现出来。一些常用的项目计划书包括：项目开发计划、验收计划、质量计划、维护计划、配置管理计划、人员计

划等。

软件工程对项目的要求是计划在先、实施在后，这意味着：项目初期拟定的计划将会成为项目开展的驱动或指南。

### 3.4.1 项目开发计划

当项目经过可行性评估获得通过以后，接着就应该编制项目开发计划，其涉及的内容包括：

- (1) 开发团队的组织结构，人员组成与分工。
- (2) 项目成本预算。
- (3) 项目对硬件、软件的资源需求。
- (4) 项目任务分解和每项的任务里程碑标志。
- (5) 基于里程碑的进度计划和人员配备计划。
- (6) 项目风险计划。
- (7) 项目监督计划。

项目开发计划需要给出影响软件项目的各项约束条件，例如：项目交付期限、现有的人员情况、项目总体预算等。但是，项目初期的问题往往是比较模糊的，项目进行过程中也有可能出现一些没有预料到的问题。因此，制定开发计划时应该有一定的灵活性，以保证当项目进行过程中出现偶然事件或没有完全按计划达到预期目标时，能够对计划做适当的调整。

当需要调整的计划 and 用户所提出的要求发生矛盾时，就不得不得和用户进行协商了。为了避免或减少出现这些问题，有经验的项目管理者在项目初期制定计划时往往会保持一种低调，他会把各种不利因素尽量考虑进去。

### 3.4.2 项目进度表

项目进度是基于里程碑制定的，例如，可行性分析、需求分析、概要设计等。每一个里程碑都将产生明确的结果导出。通过里程碑来管理项目进度，其好处是能够使对项目的管理落实到项目过程中的每一个关键点上去，由此达到对项目的微观管理的目的。

在安排项目进度时，需要估算完成各项活动所需的时间和资源，并按照一定的顺序把它们严密地组织起来。除非进行进度安排的项目与原来的项目相似，否则新的项目进度不能沿用原来的安排。由于不同的项目可能使用不同的设计方法和实现语言，使得对资源和时间的估算更为复杂。

项目进度需要把项目中的所有工作分解为若干个独立的活动，并需要判断完成这些活动所需的时间。通常，有些活动需要并行进行，以保证人力资源得到充分利用。在安排进度时，关键任务要重点考虑，要避免因某项关键任务没有完成而使整个项目延期交付的情形出现。

正常情况下项目的各项活动应该至少持续一个星期。因此，项目进度中的活动可以以“周”为单位计量，更细的计量划分则意味着在项目进度的估算和进度表的修订上会花掉太多时间。

项目进行过程中难免有偶然事件发生。例如，做这个项目的个别人员可能生病或离职，设备可能会出故障，按计划需要配备的一些基本支持软件或硬件有可能没有按时配备等。如果这是个新项目并且技术先进，还有可能遇到没有预计到的技术困难。因此，在估算进度时，需要有一定的时间宽松。各项活动之间还应该短暂的间隔，以利于活动的过渡。

可以使用进度图表来描述项目进度。其中，甘特图表是一种比较常用的进度图表，可以直观地描述项目任务的活动分解，以及活动之间的依赖关系、资源配置情况、各项活动的进展情况等。

甘特图表使用条形图表示每项活动的开始、结束时间以及活动进展，通过链接的箭头线可以表达活动之间的依赖关系，还可以通过表格对每项活动进行详细的描述。例如图 3-4 中的甘特图，能够清楚地展现某“人力资源管理系统”开发过程中各项活动的时间安排，活动之间的依赖关系，以及各项活动的进展情况。

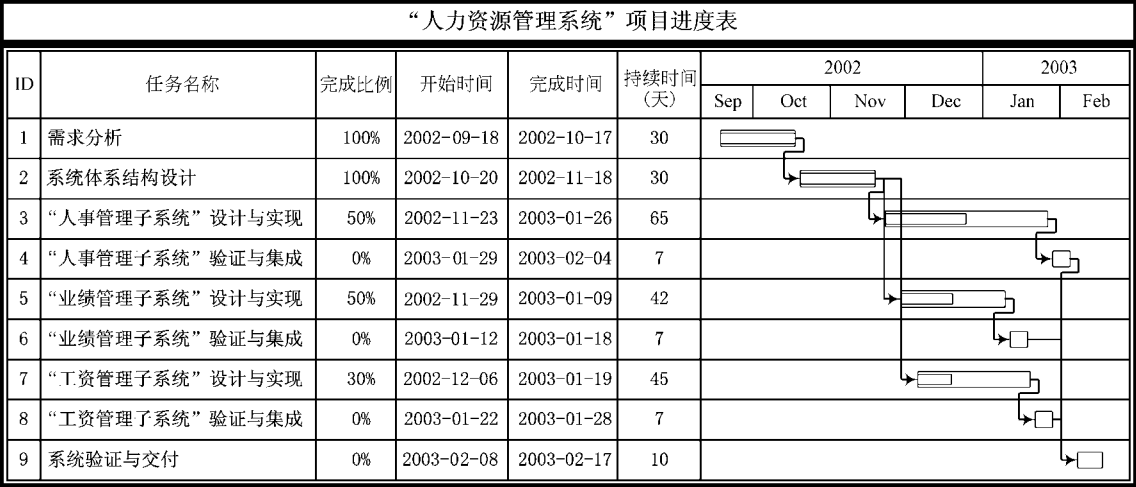


图 3-4 使用“甘特图表”描述项目进度

小 结

1. 计算机系统分析

(1) 计算机系统

计算机系统是一个非常复杂并具有智能特性的开发系统，包括：硬件系统、软件系统、网络通信系统、人工操作系统等诸多子系统。

(2) 系统分析

系统分析是对软件项目的高层分析，需要获取的是有关系统的框架描述，并需要使系统从它所处的环境中分离出来，为划分系统边界与确定系统构架提供依据。

(3) 系统分析模型

分析模型是指采用作图方式对系统进行直观的描述。系统前期分析过程中经常使用的图形模型有系统框架图和系统流程图。其中，系统框架图用于说明系统的基本构造框架，而系统流程图则用于表现系统的基本加工流程。

2. 项目可行性分析

(1) 意义

- 以少量的费用对项目能否实施尽早作出决断。
- 根据项目条件限制，对系统的体系构造、工作模式等作出高层抉择。
- 其结果可作为一个高层框架被用于需求分析之中。

#### (2) 分析内容

- 技术可行性：从技术与技术资源这两个方面作出可行性评估。
- 经济可行性：从项目投资和经济效益这两个方面作出可行性评估。
- 应用可行性：从法律法规、用户操作规程等方面作出可行性评估。

#### (3) 分析过程

- 建立系统模型。
- 进行可行性评估。
- 撰写可行性研究报告。

#### 3. 项目成本效益分析

(1) 项目成本估算方法：基于软件规模的成本估算；基于任务分解的成本估算。

(2) 项目效益分析指标：纯收入；投资回收期；投资回收率。

#### 4. 项目规划

##### (1) 项目开发计划

项目开发计划涉及的内容包括：

- 开发团队的组织结构，人员组成与分工。
- 项目成本预算。
- 项目对硬件、软件的资源需求。
- 项目任务分解和每项的任务里程碑标志。
- 基于里程碑的进度计划和人员配备计划。
- 项目风险计划。
- 项目监督计划。

##### (2) 项目进度表

项目进度是基于里程碑制定的，可以使用进度图表来描述项目进度。甘特图表是一种常用的项目进度图表，可以直观地描述项目任务的活动分解，以及活动之间的依赖关系、资源配置情况、各项活动的进展情况等。

## 习 题

1. 计算机系统有哪些特点？

2. 图 3-5 是某“零配件仓库管理系统”的系统流程图。从图中可以看出该系统的以下特点：

(1) 系统将以入库单、出库单为依据进行入出库操作，并通过更新库存程序对库存记录文件进行更新，通过入出库记录程序将入库、出库记录添加到入出库数据表中。

(2) 系统将以库存数据表中记录的库存量为依据确定采购计划，当某零配件的库存量低于规定的下阈值时，该零配件将会进入到订购单中。其中，订购单由生成采购计划程序产生。

(3) 系统将以入出库数据表中的记录为依据进行入出库统计分析，产生入出库统计分析报表。

试根据对图 3-5 中的系统流程图的理解，将“零配件仓库管理系统”从其工作环境中分离出来，并建立

该系统的“系统框架图”。

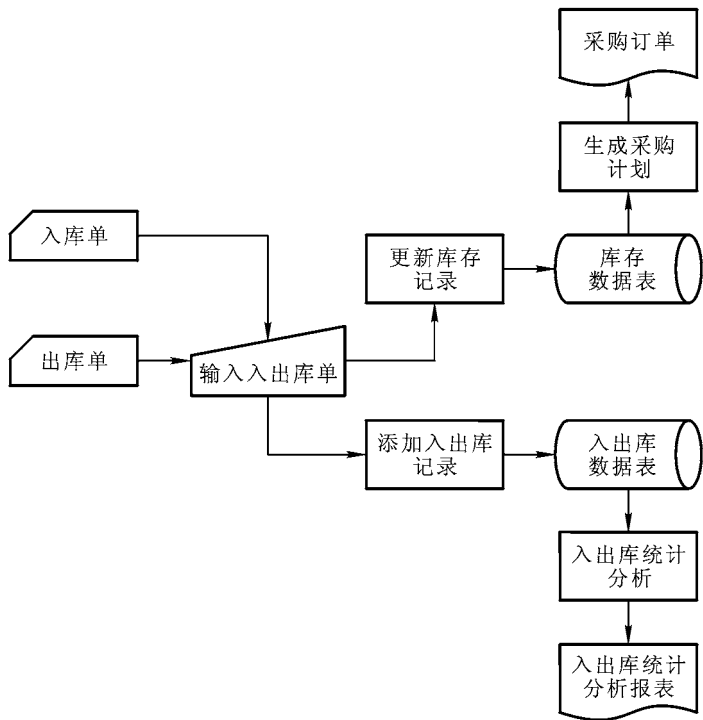


图 3-5 “零配件仓库管理系统”的系统流程图

3．可行性分析的意义主要体现在哪些方面？

4．技术可行性评估主要涉及哪些方面的问题？

5．某软件项目已将有待开发的软件分解为 600 个对象点数，其中有 20 %可以复用。假设该项目参入人员月平均工资为 3 000 元人民币，每月按 20 个工作日计算，每人每天能够完成 1 个对象点数的工作量。试估算该软件项目的人力成本。

6．假设某软件项目计划按照瀑布模型实施开发，并计划在 7 个月内完成开发任务。其任务进度安排是：需求分析 1 个月、软件设计 2 个月、编码与单元测试 3 个月、系统集成 1 个月。试使用甘特图表画出该软件项目的任务进度图表。



## 第 4 章 软件需求分析

软件需要解决的是用户所面临的现实问题，但是，这些现实问题需要由软件技术人员来解决。情况往往是，开发软件的技术人员精通计算机技术，但并不熟悉用户的业务领域；而用户清楚自己的业务，却又不太懂计算机技术。因此，对于同一个问题，技术人员和用户之间可能存在认识上的差异。也因此，在软件技术人员着手设计软件之前，需要由既精通计算机技术又熟悉用户应用领域的软件系统分析人员，对软件问题进行细致的需求分析。

需求分析是软件工程过程中一个重要的里程碑。在需求分析过程中，软件系统分析人员通过研究用户在软件问题上的需求意愿，分析出软件系统在功能、性能、数据等诸多方面应该达到的目标，从而获得有关软件的需求规格定义，其信息流如图 4-1 所示。

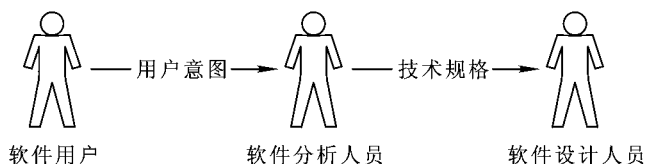


图 4-1 需求分析中的信息流

需求分析是在软件系统分析人员的操作下进行的，在这个过程中，用户和开发者之间需要达成的是对系统的一致性需求认识。实际上，可以把软件系统分析人员看成是软件用户与软件开发技术人员之间的信息通道，其作用是使用户对软件问题的现实描述，能够有效地转变为开发软件的技术人员所需要的对软件的技术描述，以方便技术人员对软件的技术构建。

### 4.1 需求分析的任务

需求分析需要实现的是将软件用户对于软件的一系列意图、想法转变为软件开发人员所需要的有关软件的技术规格，并由此实现用户和开发人员之间的有效通信，它涉及面向用户的用户需求和面向开发者的系统需求这两个方面的工作内容。

#### 4.1.1 用户需求

用户需求是用户关于软件的一系列意图、想法的集中体现，涉及软件的操作方式、界面风格、报表格式，用户机构的业务范围、工作流程，以及用户对于软件应用的发展期望等。因此，用户需求也就是用户关于软件的外界特征的规格表述。

实际上，用户需求提出了一个有关软件的基本需求框架，具有以下特点：

(1) 用户需求直接来源于用户，可以由用户主动提出，也可以通过与用户交谈，或对用户进行问卷调查等方式获取。由于用户对计算机系统认识上的不足，分析人员有义务帮助用户挖

掘需求，例如，可以使用启发的方式激发用户的需求想法。如何更有效地获取用户需求，既是一门技术，也是一门思维沟通艺术。

(2) 用户需求需要以文档的形式提供给用户审查，因此需要使用流畅的自然语言或简洁清晰的直观图表来进行表述，以方便用户的理解与确认。

(3) 可以把用户需求理解为用户对软件的合理请求，这意味着，用户需求并不是开发者对用户的盲目顺从，而是建立在开发者和用户共同讨论、相互协商的基础上。

(4) 用户需求主要是为用户方管理层撰写的，但用户方的技术代表，软件系统今后的操作者，以及开发方的高层技术人员，也有必要认真阅读用户需求文档。

#### 4.1.2 系统需求

系统需求是比用户需求更具有技术特性的需求陈述，是提供给开发者或用户方技术人员阅读的，并将作为软件开发人员设计系统的起点与基本依据。

系统需求需要对系统在功能、性能、数据等方面进行规格定义，由于自然语言随意性较大，在描述问题时容易发生歧义，因此系统需求往往要求用更加严格的形式化语言进行表述，例如 PDL 伪码，以保证系统需求表述具有一致性。

系统需求涉及有关软件的一系列技术规格，包括：功能、数据、性能、安全等诸多方面的问题。

##### 1. 功能需求

功能需求是有关软件系统的最基本的需求表述，用于说明系统应该做什么，涉及软件系统的功能特征、功能边界、输入输出接口、异常处理方法等方面的问题。也就是说，功能需求需要对软件系统的服务能力进行全面的详细的描述。

在结构化方法中，往往通过数据流图来说明系统对数据的加工过程，它能够在一定程度上表现出系统的功能动态特征。也就是说，可以使用数据流图建立软件系统的功能动态模型。

##### 2. 数据需求

数据需求用于对系统中的数据，包括：输入数据、输出数据、加工中的数据、保存在存储设备上的数据等，进行详细的用途说明与规格定义。

在结构化方法中，往往使用数据字典对数据进行全面准确的定义，例如，数据的名称、别名、组成元素、出现的位置、出现的频率等。当所要开发的软件系统涉及到对数据库的操作时，还可以使用数据关系模型图（ER 图，详见 4.4.3 中介绍）对数据库中的数据实体、数据实体之间的关系等进行描述。

##### 3. 其他需求

其他需求是指系统在性能、安全、界面等方面需要达到的要求。

### 4.2 需求分析过程

需求分析是对软件系统的后期分析，需要进行一系列的活动，包括：分析用户需求、建立需求原型、分析系统需求和进行需求验证等，其活动流程如图 4-2 所示。

从图 4-2 可以看到需求分析的以下一些特点：

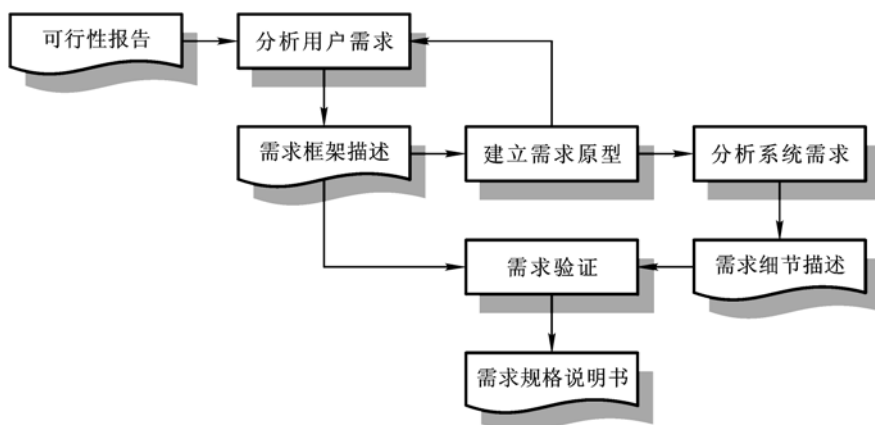


图 4-2 需求分析过程

(1) 需求分析可以可行性分析中软件系统的高层模型为前提，首先需要进行的是分析用户需求，由此可以提出关于软件的需求框架。

(2) 由于用户对计算机信息系统认识上的模糊性，以致直接来源于用户的需求想法往往存在着许多缺陷，例如：需求冲突、需求遗漏。为提高用户需求的有效性，可以按照需求框架的要求建立需求原型让用户确认，然后再通过需求原型去提取系统模型。

(3) 系统需求是面向技术人员的。因此，它不仅需要从需求原型中提取系统模型，而且需要对系统模型进行细节化处理，例如：数据流细化、数据字典分解、类模型的定义等，由此获得对系统的详细的技术性需求说明。

(4) 对需求框架、系统模型和需求细节等文档进行有效性验证，由此产生出用户方、开发方都能接受的关于软件的需求规格说明书。

## 4.3 用户需求获取

优秀软件总是能够最大限度地满足用户需求。因此，有效获取用户需求，是实施软件开发时需要完成的第一项工作。

### 4.3.1 研究用户

在建立软件抽象模型过程中，用户可以被当作一个抽象概念，泛指诸多从系统获得服务的系统以外的对象，例如：软件使用机构，软件操作人员，以及从软件获得信息服务的其他系统或设备。也就是说，可以把用户看作为系统的外部应用接口。但从软件的商业服务角度来看，用户则主要是指购置软件的机构、使用软件的部门、操作软件的人。

软件是为用户开发的，为了使软件能够更好地为用户服务，在软件需求阶段，软件开发者应该尽量充分地 and 用户进行沟通，了解用户的意图。例如，用户希望软件为他提供哪些方面的服务，用户在操作软件时有哪些工作习惯，喜欢什么样的工作界面等。

软件往往需要为各种不同的用户服务，例如，用户机构负责人、使用软件的部门主管、软件操作人员，他们都是用户，但由于所处位置不同，因此会有不同的需求。软件操作人员大多

需要较长时间地操作软件，因此他们会特别关心软件的工作方式、界面布局；使用软件的部门主管则一般不会直接操作软件，但他需要从软件中得到年度报表之类的打印数据，因此比较关心软件能够提供哪些方面的报表，报表格式如何等。

用户情况是复杂的，为了更好地研究用户，有必要对软件用户做一些分类，例如：按软件需求层次不同，将用户分为高层用户、中层用户和基层用户；按使用软件时间长短不同，将用户分为熟练用户和非熟练用户；按是否直接操作软件系统，将用户分为直接用户和间接用户。

### 4.3.2 从调查中获取用户需求

直到今天，用户调查仍是最基本的用户需求信息收集方法。应该说，用户需求信息来源是多方面的，为了保证用户需求的完整性，在需求分析过程中，软件系统分析人员往往需要调查各类能够代表用户意愿的相关人员，如图 4-3 所示。

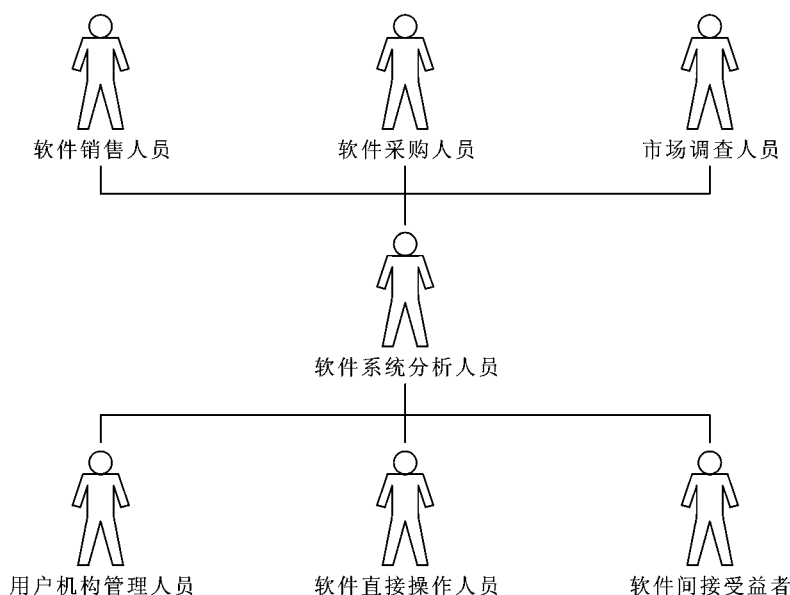


图 4-3 分析人员应该从各个方面调查用户

实际上，针对不同的应用项目通常会有不同的调查内容，需要采用不同的调查策略。例如，开发一个信息管理系统，其用户需求调查一般会涉及以下方面的内容：

（1）调查用户的组织机构，包括：该组织的部门组成，各部门的职责等。由此得到的调查结果将作为分析软件系统领域边界的基本依据。

（2）调查用户各部门的业务活动，包括：各个部门输入和使用什么数据，如何加工处理这些数据，输出什么数据，输出到什么部门等。这是需求调查的重点内容，其结果将作为分析软件系统工作流程的基本依据。

（3）调查用户对软件系统的各项具体要求，包括：数据格式、操作方式、工作性能、安全保密等方面的要求。

在调查过程中，可以根据不同的问题和条件，使用不同的调查方法。比较常用的调查方法有以下几种：

#### 1. 访谈用户

访谈用户就是面对面地跟单个用户进行对话。例如，请用户机构高层人员介绍用户的组织结构、业务范围、对软件应用的期望。

#### 2. 开座谈会

当需要对用户机构的诸多部门进行业务活动调查与商讨时，可以考虑采用开一个座谈会的方式。这既有利于节约调查时间，又能使各部门之间就业务问题进行协商，以方便对与软件有关的业务进行合理分配与定位。

#### 3. 问卷调查

问卷调查一般是通过精心设计的调查表去调查用户对软件的看法。当面对一个庞大的用户群体时，可能需要采用问卷调查形式进行用户调查。例如在开发通用软件时，为了获得广大用户对软件的看法，就不得不采取问卷方式。如果调查表设计得合理，这种方法很有效，也易于为用户接受。

#### 4. 跟班作业

跟班作业就是软件分析人员亲身参加用户单位业务工作，由此可直接体验用户的业务活动情况。这种方法可以更加准确地理解用户的需求，但比较耗费时间。

#### 5. 收集用户资料

尽管有待开发的软件需要在较长时间以后才能交付用户使用，但跟软件有关的工作用户却一直在以其他方式或通过其他系统进行着，并且也一直在产生结果。用户资料主要就是指这些结果，例如：年度汇总报表、提货单、工资表等。软件分析人员应该认真收集这些资料，由此可以更加清楚地认识用户的软件需求。

### 4.3.3 通过原型完善用户需求

需求原型可用来收集用户需求，对用户需求进行验证，由此可帮助用户克服对软件需求的模糊认识，并让用户需求能够更加完整地得以表达。例如，用户对软件应该提出哪些方面的服务，操作界面应该如何等可能拿不定主意，为了使用户能够更加直观地表述自己的需求意愿，可以先构造一个原型给用户体验。

原型需要根据用户的评价而不断修正，这也有利于挖掘用户的一些潜在需求，使得用户需求能够更加完整地得以表达。一般情况下，开发人员将软件系统中最能够被用户直接感受的那一部分东西构造成为原型。例如，界面、报表或数据查询结果。实际上，在诸多原型中，界面原型是应用得最广泛的原型。

如图 4-4 所示，需求原型可以建立在用户所提出的需求框架基础上。需求原型的作用是能够方便系统模型的创建。也就是说，需求原型可以方便由用户需求到系统需求的过渡。

### 4.3.4 用户需求陈述

用户需求陈述用于记录用户需求，为了方便用户理解，一般采用自然语言风格进行描述。以下是关于某商品仓库管理系统的用户需求陈述。

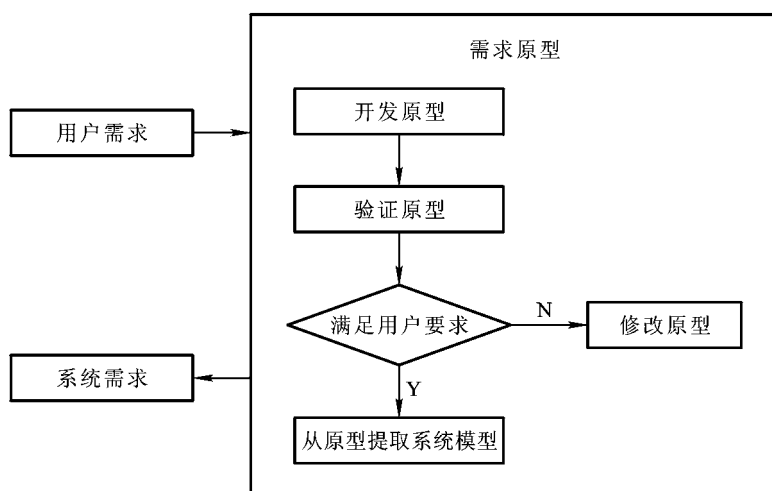


图 4-4 需求原型

（1）仓库管理系统将被计划部门、仓库管理部门、采购部门、销售部门的相关工作人员使用。其中，计划部门制定商品计划，涉及：设置商品类别、登记商品、制定商品报损计划和进行商品流通数据汇总；仓库管理部门完成仓库的日常管理，涉及：商品入库、出库、报损和查询等多项操作；采购部门可以查询商品库存情况、打印商品定货报表；销售部门可以查询商品库存情况和提交商品定货请求。

（2）由于不同部门有不同的任务，因此系统需要提供针对部门的权限管理机制和针对工作人员的登录注册机制。系统将通过一位系统管理员进行部门授权与工作人员注册管理。

（3）进入仓库管理系统的工作人员需要有惟一的个人身份标识，它既是工作人员登录系统时的身份验证依据，也是工作人员在进行商品操作时的经手人标记。尽管工作人员的姓名也可以用做其身份标识，但不同的工作人员有可能会出现姓名重复，因此有必要为工作人员设置一个专门的身份标识码。

（4）仓库以商品品种为基本单位进行管理，所有商品都要由计划部门按品种进行登记，涉及商品编码、名称、类别、库存下限值等数据。其中，商品库存量初始值为零。

（5）仓库商品涉及入库、出库、报损这三种流通方式。凭采购部门填写的入库单进库，凭销售部门填写的出库单出库，凭计划部门制定的报损计划报损。商品的任何流通都需要以流水方式记录到商品流通表中，并对商品库存量进行更新。当商品出库、报损时，必须考虑到该商品的当前库存量是否能够满足操作需要。出库、报损后，若商品库存量低于库存下限值，将自动产生定货请求。

（6）可以按商品类别或品种进行商品库存情况和当月商品流通情况的查询。另外，仓库管理系统需要自动在月底对商品流通数据进行盘查，包括：按月打印商品流通分类汇总报表，并按月备份商品流通数据，然后将已备份的商品流通数据进行合计整理。

为了方便用户对有待开发系统的认识与评价，可以使用后面 4.4.1 节中的功能层次图来描述系统的业务级功能组织结构。

## 4.4 结构化分析建模

人在求解问题时，首要需要做的是理解问题，并且对问题理解得越透彻，则这个问题就越容易解决。所谓模型，就是为了理解问题而对问题做的一种符号抽象。可以把模型看作为一种思维工具，利用这种工具可以把问题规范地表示出来。

模型一般由一组图示符号和组织这些符号的规则组成。因此，分析时期的建模，就是针对用户需求、系统需求等，采用图示方式进行直观描述。

软件问题往往是复杂的，而建模可以使问题简化。人的头脑每次只能处理一定数量的信息，模型通过把系统分解成人的头脑一次能处理的若干个子部分，从而减少系统的复杂程度。分析时期建立软件模型的作用是多方面的，可以通过模型实现由用户需求向系统需求的过渡，并可通过模型获得对系统需求的更具细节性的推论。实际上，分析时期产生的模型还可以被引用到系统设计中，作为设计前导。

为了开发复杂的软件系统，往往需要从不同角度去构造系统模型。例如，用于描述系统功能组织结构的层次模型，用于描述系统中数据加工流程的数据流模型，用于描述数据实体及其关系的数据关系模型，用于描述系统行为过程的系统状态模型等。

本节将要介绍的只是传统的结构化分析建模方法（SA）。有关面向对象分析建模方法（OOA）将在第6章中介绍。

### 4.4.1 功能层次模型

功能层次模型图使用矩形来表示系统中的子系统或功能模块，使用树形连线结构来表达系统所具有的功能层级关系。

图4-5所示为前面4.3.4节所描述的“商品仓库管理系统”的功能层次图。由图可以看到，层次图能够方便地描述系统的组成关系。由于功能层次图的简洁性，特别适合就系统的功能组织结构方面的问题跟用户进行讨论。

实际上，不仅可以使用层次图清晰地说明系统的功能组成关系，也可以使用功能层次图对系统的功能关系进行调整，从而使系统的诸多功能得到更加合理地分配。

### 4.4.2 数据流模型（DFD图）

#### 1. 数据流图的特点

数据流模型由DeMarco于1978年提出，并随着他的结构化分析思想一起被广为流行。数据流图用于描述系统对数据的加工过程，其图形符号是一些具有抽象意义的逻辑符号。表4-1所列是数据流图的基本符号。

在软件定义时期，数据流图被用来描述系统的逻辑加工步骤。由于数据流图能够为有待开发的系统提供一种简洁的逻辑图形说明，能够方便用户对系统分析的理解，因此，它也被用做开发者和用户之间的信息交流工具。

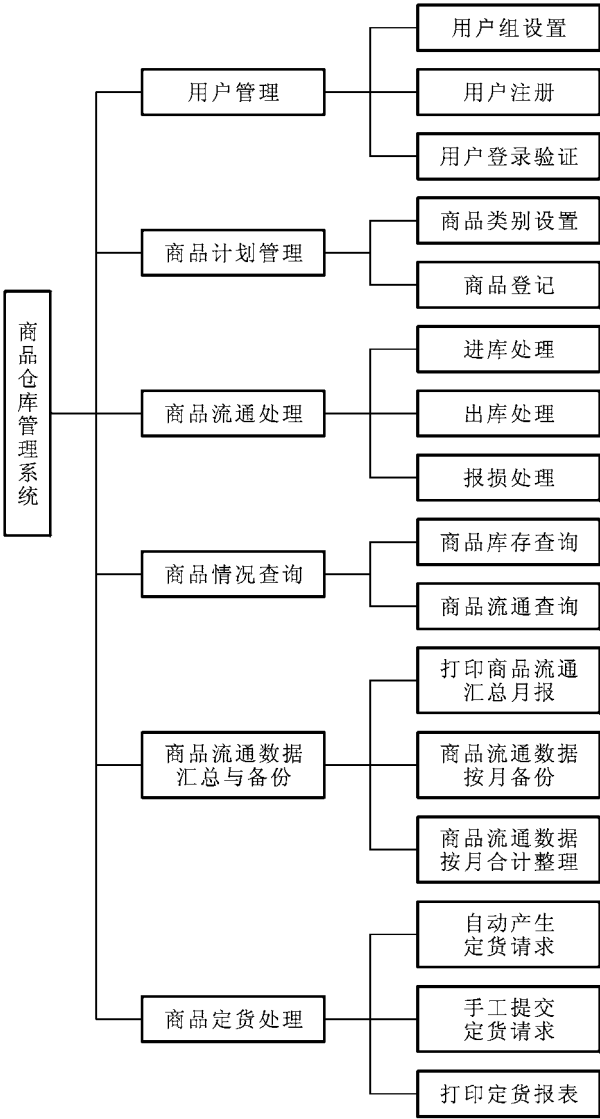






图 4-5 商品仓库管理系统层次图

表 4-1 数据流图的基本图形符号

图 形 符 号	说 明
	数据接口，系统的外部源头或终点，用来表示系统与外部环境的关系。可以将接口理解为系统的服务对象，例如：系统的操作人员，使用系统的机构或部门，系统之外的其他系统或设备等
	数据处理，表示将数据由一种形式转换成了另一种形式的某种活动。数据处理框上必须有数据的流入与流出，用以描述流入处理框的数据经过处理变换成了流出的数据。对数据的处理可以是程序处理、人工处理、设备处理等



续表

图 形 符 号	说 明
	数据存储, 数据的静态形式, 用来表示任何对数据的存储。例如, 用于临时存储的内存变量, 存储在磁盘或磁带上的数据文件、数据表、记录集, 存储在纸质上的数据备份等。它可以是介质上某个存储单元的全部数据内容, 也可以是介质上某个存储单元的存储片段
	数据流, 图中数据的动态形式, 表示数据的流向。数据流必须与一个数据处理相连接, 以表示数据处理在接收或发送数据的过程中给数据带来的变换。可以通过数据流将某个数据处理连接到其他的数据处理, 或连接到数据存储、数据接口

## 2. 数据流图的用途

可以依靠数据流图来实现从用户需求到系统需求的过渡。例如, 可以将用户需求陈述中的关键词、动词提取出来, 其中的名词可以作为数据流图中数据源、数据存储, 而动词则可以作为数据流图中的数据加工进程。

数据流图也能够方便系统物理模型与逻辑模型之间的转换, 可以将 3.1.3 节中系统流程图经过符号转换而获得系统的数据流图, 例如图 3-3 的系统流程图, 通过转换符号可以得出图 4-6 的数据流图。数据流图的这个特点表明, 可以基于系统的基本物理框架而抽取它的逻辑模型。

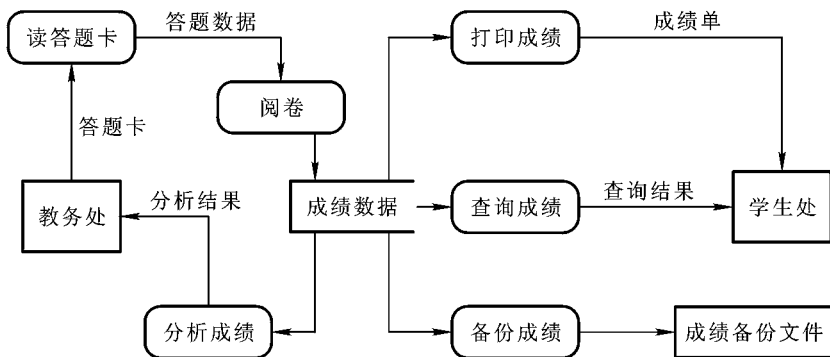


图 4-6 “自动阅卷系统”数据流图

软件系统是复杂的, 为了方便问题的解决, 有必要将系统进行分解, 由此将一个大的复杂问题解剖为许多小的相对简单的问题。例如, 可以按照系统的功能构成, 将系统分解为许多子系统, 各子系统又可以再分解为许多更小的功能模块, 由此可以不断地了解软件系统的功能细节。由于数据流图使用的是抽象的图形符号, 因此, 它不仅能够描述系统对数据的加工步骤, 而且能够依靠对其图形符号的逻辑细化而方便地实现对系统中数据加工步骤的有效分解。

## 3. 数据流细化过程

数据流细化的过程即是从上至下对系统功能进行分层描述的过程, 如图 4-7 所示。其中, 高层数据流对功能的描述是抽象的, 但通过逻辑细化能够深入到系统内部的低层数据流, 而使得对功能的描述逐步具体化。结构化分析就是基于数据流的细化实现的, 它是结构化分析方法的关键。

实际上, 数据流图对系统的描述可以从任何层面开始, 只要那个层面的诸多软件问题是清晰的, 则在该层面上获得的数据流图也就可以是清晰的。但是, 进入系统的层面越深, 则遇到

的问题点越多，数据流越复杂。为了更加清楚地表现系统的功能，数据流图往往从容易辨别的高层开始，然后通过数据流的细化逐步深入。这个过程也就是从抽象到具体的解决问题方法在软件分析上的具体体现。

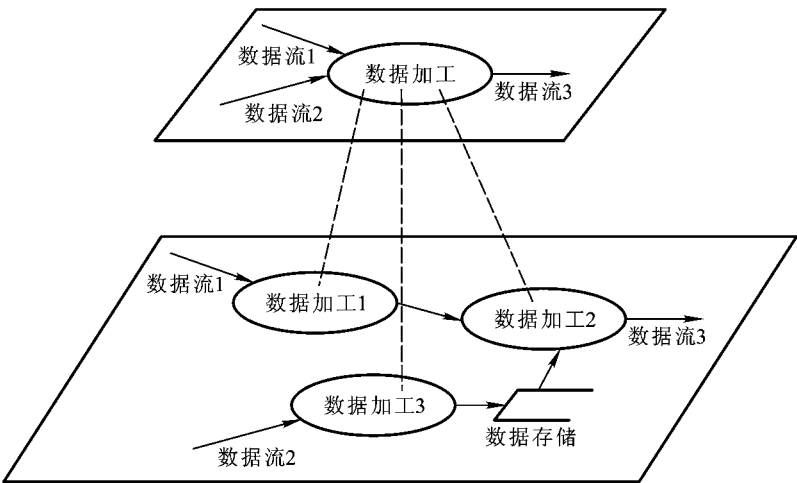


图 4-7 数据流图对系统的分层描述

当面对一个有待开发的新系统进行数据流描述时，数据流图往往需要从最顶层画起，使用一个数据处理框来表示整个系统，以反映系统与周围环境的关系，然后通过数据流的细化而获得 0 层、1 层，以及更下层的数据流图。

图 4-8 是一个“工资管理系统”的顶层数据流图，其中的处理框表示所要描述的系统，而三个外部接口（人事处、财务处、员工所在工作部门）则可以表示系统的工作环境。系统与外部接口之间的通信可以通过数据流，图 4-8 中有四个数据流，即：职工清单、档案工资、业绩工资、工资报表。

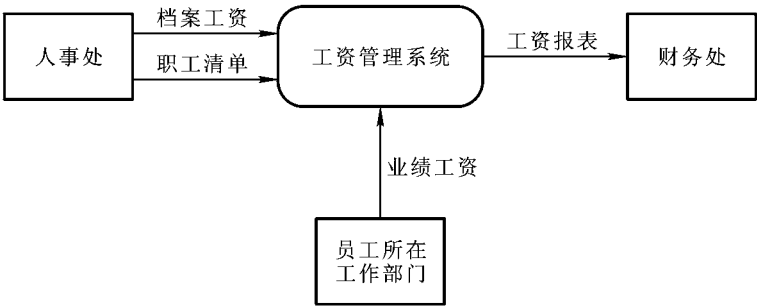


图 4-8 “工资管理系统”的顶层数据流图

当需要对高层数据流细化以获得对低层数据流的描述时，一种有效的方法是对高层数据流图中的数据加工进行合理分解。通过把一个数据加工分解为多个数据加工可以看到这个数据加工的内部细节。

例如，图 4-8 所描述的“工资管理系统”数据流图。假如“工资管理系统”可以具有以下

三项功能：（1）输入职工名册清单；（2）从员工的档案工资和业绩工资的计算中产生工资数据；（3）依据人事部门提供的职工清单按月打印出员工的工资报表。那么，可以考虑将“工资管理系统”分解为以下三项处理，即：（1）提供职工清单；（2）产生工资数据；（3）打印工资报表。

图 4-9 即是依照上述功能分解而从顶层数据流图细化出来的 0 层数据流图。从 0 层数据流图可以看到数据流在细化时具有以下特点：

- （1）与上一层数据处理“工资管理系统”相关的数据流被继承了下来。
- （2）上一层数据处理“工资管理系统”中不可见的内部数据流变成了可见的外部数据流。

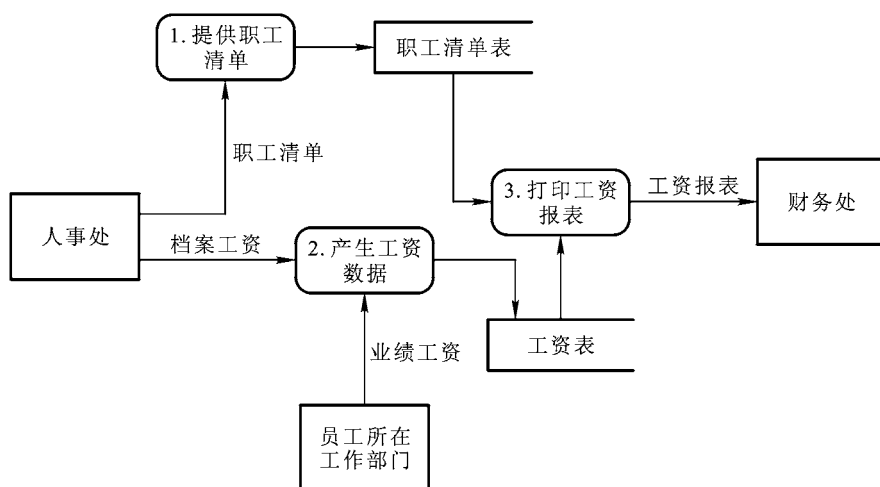


图 4-9 “工资管理系统”的 0 层数据流图

数据流细化被用来分析系统的内部功能构造。然而，面对一个具有一定规模的软件系统，0 层数据流图往往只能对其功能进行一般性的高层描述。因此，为了使数据流对系统功能的描述更加具体，数据流细化往往还需要继续深入下去。

实际上，可以使用数据流进行软件结构的映射（结构化设计）。一般情况下，假如希望将数据流用于软件设计，则对数据流细化更需要以设计中的模块构件作为分解目标。

因此，可以考虑对“工资管理系统”进行更加深入的数据流细化。例如，“工资管理系统”0 层数据流图中的“产生工资数据”处理，假如其工资数据的产生涉及数据录入、数据计算等更加具体的数据操作，则“产生工资数据”处理可以进一步分解为：“录入档案工资”、“录入业绩工资”、“计算工资”这三项处理。

图 4-10 即是对“产生工资数据”处理框进一步细化后产生出来的 1 层数据流图。其中的数据加工流程是：

（1）来源于“人事处”和“员工所在工作部门”的“档案工资”、“业绩工资”数据流，经“录入档案工资”、“录入业绩工资”的处理之后，被分别存入到“档案工资表”、“业绩工资表”这两个数据存储之中。

（2）系统可以从“档案工资表”和“业绩工资表”这两个数据存储读取数据，然后通过“计

算工资”的处理产生出“工资数据”数据流。这个数据流将被存入到“工资数据表”中。

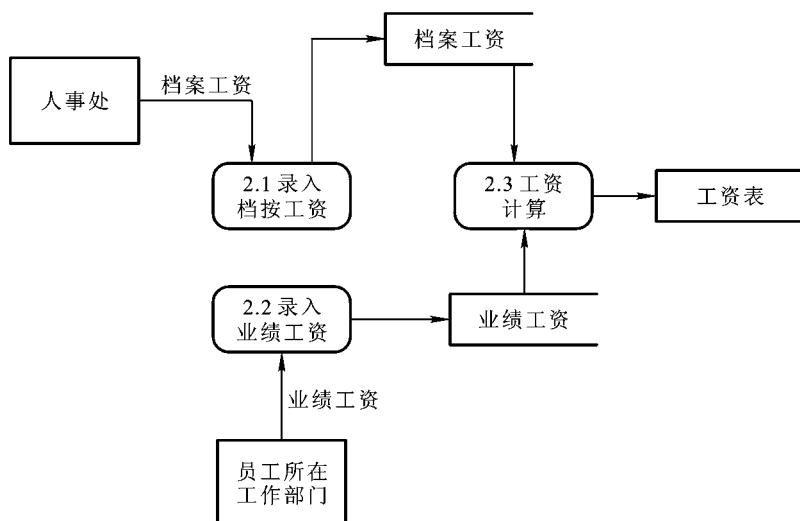


图 4-10 对“产生工资数据”处理框进一步细化后的 1 层数据流图

显然，这又是一个既涉及细化又包含继承的分析过程。与“工资管理系统”直接相关的数据流成分被继承了下来（人事处、员工所在工作部门、工资数据表），而图 4-9 中不可见的内部数据流则经过细化变成了可见的外部数据流。

#### 4. 数据流图中符号的命名

数据流图中的图形符号一般都需要命名，并遵循以下命名规则：

（1）数据接口：使用名词或名词短语命名。例如：人事处、财务处、工资数据录入员、系统管理员、读卡设备、打印设备。

（2）数据存储：使用名词或名词短语命名。例如“工资数据表”。当数据存储是存储介质上某个存储单元的存储片段时，其名称还需要用到限定词，例如“在职人员档案工资”。

（3）数据流：使用名词或名词短语命名。但为了提高数据流图的清晰度，从数据存储中流出，或流入数据存储的数据流，在不会发生名称混淆的前提下，可以省略名称。例如图 4-9 中从“档案工资表”、“业绩工资表”流出，或流向“档案工资表”、“业绩工资表”的数据流。

（4）数据处理：数据处理涉及处理方式与处理对象两方面的内容，一般使用“动词+名词短语”的动宾结构来进行命名。例如，“录入档案工资”、“打印工资报表”。由于对数据流的细化是通过对数据处理的分解实现的，考虑到对细化后的数据流图进行分层检索的便利，可以对处理框进行合适的数字编码。例如，“2. 产生工资数据”、“2.1 录入档案工资”、“2.2 录入业绩工资”、“2.3 计算工资”。

#### 5. 数据流图中的数据字典

在需求分析中，数据字典是各类数据描述的集合，能够提供对数据的详细规格定义，并可用于验证数据，以发现系统在数据需求描述中是否出现遗漏。

数据流图中的数据字典能够提供对图中的诸多数据元素的更加详细的说明。其一般要求

是：(1) 对数据的定义应该是严密、精确、一致的，不能有二义性；(2) 需要对数据流图中的每一个被命名的数据元素进行定义；(3) 需要分类定义各种不同种类的数据元素，或采用类别代号加以区别。

数据流图中的数据字典通常包括数据项、数据结构、数据流、数据存储、数据接口和数据处理过程这几个部分的数据内容。其中，数据项是数据的最小组成单位，若干个数据项可以组成一个数据结构。数据字典就是通过对数据项和数据结构的定义来描述数据流、数据存储的逻辑内容的。

#### (1) 数据项

数据项是不可再分的数据单位。对数据项的描述通常包括以下内容：

{ 数据项名，数据项含义说明，别名，数据类型，长度，取值范围，取值含义，与其他数据项的逻辑关系 }

#### (2) 数据结构

数据结构反映了数据之间的组合关系。一个数据结构可以由若干个数据项组成，也可以由若干个数据结构组成，或由若干个数据项和数据结构混合组成。对数据结构的描述通常包括以下内容：

{ 数据结构名，含义说明，组成：{ 数据项或数据结构 } }

在定义数据结构时，可以采用以下符号说明数据的组成：

= 被定义为，表示数据组成。

+ 与，用于连接两个数据分量。

[...|...] 或，从若干数据分量中选择一个，方括号中的数据分量用“|”号隔开。

m{...}n 重复，重复花括号内的数据，最少重复 m 次，最多重复 n 次。

(...) 可选，圆括号内数据可有可无。

#### (3) 数据流

数据流是数据结构在软件系统内传输的路径。对数据流的描述通常包括以下内容：

{ 数据流名，说明，数据流来源，数据流去向，组成 { 数据结构 }，平均流量，高峰期流量 }

其中，数据流来源是说明该数据流来自哪个过程。数据流去向是说明该数据流将到哪个过程去。平均流量则是指在单位时间（每天、每周、每月等）里的传输次数。高峰期流量则是指在高峰时期的数据流量。

#### (4) 数据存储

数据存储是数据结构停留或保存的地方，也是数据流的来源和去向之一。对数据存储的描述通常包括以下内容：

{ 数据存储名，说明，编号，流入的数据流，流出的数据流，组成 { 数据结构 }，数据量，存取方式 }

其中，数据量是指每次存取多少数据，每天（或每小时、每周等）存取几次等信息。存取方式则包括：是批处理还是联机处理，是检索还是更新，是顺序检索还是随机检索等。另外，流入的数据流要指出其来源，流出的数据流要指出其去向。

可以使用表格将数据字典分类列出。例如前面介绍的“工资管理系统”数据流图中的数据字典，即可以通过表 4-2、表 4-3、表 4-4、表 4-5 列出。

表 4-2 “工资管理系统”中的数据接口

名 称	描 述	所连接的数据流
人事处	通过职工档案向系统输入职工档案工资数据	职工清单、档案工资
员工所在工作部门	通过职工业绩表现向系统输入职工业绩工资数据	业绩工资
财务处	按月从系统获取职工工资打印报表	工资报表

表 4-3 “工资管理系统”中的数据流

名 称	描 述	来 源	去 向	组 成
职工清单	由人事处依据职工档案提供的职工名册	人事处	系统	职工编号+姓名+部门
档案工资	由人事处依据职工档案提供的工资数据	人事处	系统	职工编号+档案工资金额
业绩工资	由员工所在工作部门依据职工业绩表现按月输入的工资数据	员工所在工作部门	系统	职工编号+月份+业绩工资金额
工资报表	按月从系统产生,作为发放职工工资的依据	系统	财务处	职工编号+姓名+部门+月份+档案工资金额+业绩工资金额

表 4-4 “工资管理系统”中的数据存储

名 称	描 述	输 入	输 出	组 成
职工清单表	储存由人事处提供的职工名册数据,用于按月打印工资报表	1. 提供职工清单	3.. 打印工资报表	职工编号+姓名+部门
工资表	储存经计算产生的工资数据,以用于按月打印工资报表	2. 产生工资数据	3.. 打印工资报表	职工编号+月份+档案工资金额+业绩工资金额
档案工资表	储存由人事处提供的档案工资数据,用做工资计算	2.1. 录入档案工资	2.3. 计算工资	职工编号+档案工资金额
业绩工资表	储存由员工所在部门输入的业绩工资数据,用做工资计算	2.2. 录入档案工资	2.3. 计算工资	职工编号+月份+业绩工资金额

表 4-5 “工资管理系统”中的数据元素

名 称	类 型	长度(字节)	限 制
职工编号	字符串	6	6{字符}6
姓名	字符串	8	2{字符}8
部门	字符串	10	1{字符}10
月份	整数	1	>=1 and <=12
档案工资金额	浮点数	4	两位小数
业绩工资金额	浮点数	4	两位小数

#### 4.4.3 数据关系模型(ER图)

许多应用软件系统往往需要通过数据库来存储数据。从结构上来看,数据库系统是独立于软件系统之外的专门系统,因此,在系统建模过程中,数据库需要进行专门的分析与设计。其中,需求分析时期建立的数据库模型称为概念模型。

数据关系模型图也称为 ER 图，是应用最广泛的数据库分析建模工具。数据关系建模方法最早由 Chen 于 20 世纪 70 年代中期提出，它以现实数据为建模依据，通过从现实数据中抽取数据实体、数据关系和数据属性这三类图形元素，建立数据库的概念模型。

1. 数据实体

数据实体是对应用领域中现实对象的数据抽象。例如，课程教学中的教师、学生和课程这三个现实对象，就可以作为数据实体对待。

2. 数据关系

数据关系是指不同数据实体之间存在的联系，包括：一对一、一对多、多对多三种类型的关系。数据关系类型及其描述如表 4-6 所列。

表 4-6 数据关系及其描述

数据关系类型	描 述
一对一的关系 (1:1)	例如，某家公司与该公司法人代表之间的关系。该公司只能有一个法人代表
一对多的关系 (1:n)	例如，某家公司与该公司员工之间的关系。该公司可以有多个员工
多对多的关系 (n:m)	例如，学生与所学课程之间的关系。每个学生可以学习多门课程，每门课程可以被多个学生学习

3. 数据属性

数据属性是指在数据实体与数据关系上所具有的一些特征值。例如，教师的编号、姓名、性别、职称、学历，是教师实体的属性；学生的学号、姓名、性别、专业、班级，是学生实体的属性；课程的课号、课名、学分、计划课时量，是课程实体的属性。而学习的成绩，则是学习关系的属性；讲授的实际课时量，则是讲授关系的属性。

在数据关系图中，数据实体用矩形表示，数据关系用菱形表示，数据属性用椭圆形表示。其中，能够用来标记实体的关键属性则通过在属性名称上加下划线来表示。

图 4-11 是教师、学生、课程这三个实体之间存在的教学关系的数据关系图的描述。

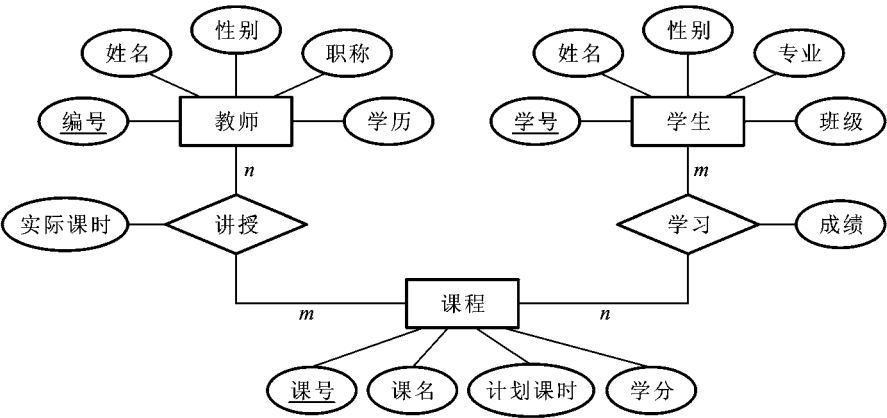


图 4-11 数据关系图

对于一些比较复杂的数据关系，为了方便分析，在画数据关系图时可以只画出实体、关系

和关键属性，而其一般属性则可以省略，例如图 4-12。

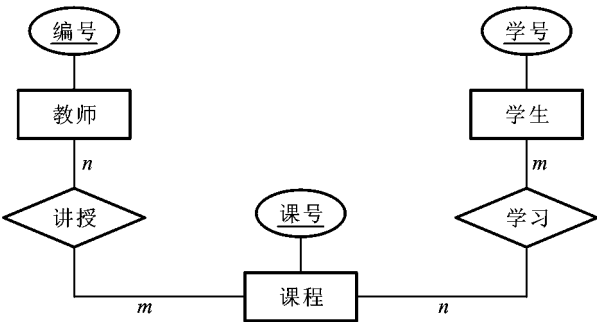


图 4-12 被简化的数据关系图

4.4.4 系统状态模型

1. 状态图的特点

状态图于 1987 年由 Harel 首先提出，其使用状态、事件等图形符号来描述系统的行为活动，用以反映系统因某个外部事件的干预而由一个可能的状态转换到另一个状态。表 4-7 所列 为状态模型图中一些常用的图形符号及其描述。

表 4-7 状态模型中的图形符号

符 号	描 述
	用来表示“初始状态”
	用来表示“最终状态”
	用来表示某个中间“状态”。可以在状态图形符号中标记状态名称、状态活动清单等。状态名称与其活动清单之间用横线 隔开
	用来表示某个中间“复合状态”，由多个子状态组成。可以在复合状态图形符号中标记状态名称、状态活动清单等。状态 名称与其活动清单之间用横线隔开
	用来表示由一个状态转换到另一个状态时将要发生的事件。在事件图形符号中，可以标记将要发生的事件名称，随事件发 生而进行状态转换时需要具备的条件，以及在状态转换时将要 产生的动作清单等
	转换分叉，用来表示由一个状态转换为两个或更多个状态
	转换连接，用来表示由两个或更多个状态转换为一个状态



<div>注释1:</div>	用于注释说明
-----------------	--------

状态模型图是通过系统的内部状态、外部事件为基本元素来描绘系统的工作流程的，这种建模方式比较适合于描述一些依赖于外部事件驱动的实时系统。

另外，状态模型也被 UML 建模语言采纳。在面向对象建模之中，状态模型可以用来对对象的状态及其变换进行细节描述。

2. 状态图的应用举例

下面是一台全自动洗衣机的大致活动过程：

（1）在洗衣机接通电源以后，洗衣机将进入待命状态。

（2）在洗衣机进入待命状态以后，操作者可以选择“设置”或“工作”。若选择“设置”，洗衣机将进入设置状态；若选择“工作”，洗衣机将进入工作状态。

（3）在洗衣机进入设置状态以后，操作者可以设置洗衣水位，洗衣机工作流程，并可在完成设置之后选择“确定”返回待命状态。

（4）在洗衣机进入工作状态以后，洗衣机将按照设置的流程进行工作。若洗衣机在洗衣过程中选择暂停，洗衣机将进入暂停等待状态；若洗衣机在洗衣过程中出现洗衣故障，洗衣机将鸣报警音，并进入故障等待状态。在洗衣机完成流程规定的工作以后，洗衣机进入结束等待状态。

（5）在洗衣机进入暂停等待状态以后，操作者可选择“恢复”，使洗衣机返回工作状态。

（6）在洗衣机进入故障等待状态以后，操作者可在排除洗衣故障之后，选择“恢复”，使洗衣机返回工作状态。

（7）在洗衣机进入结束等待状态以后，操作者可以切断电源结束洗衣过程。

根据上述活动内容，可以画出该洗衣机的状态图模型。其状态图模型如图 4-13 所示。

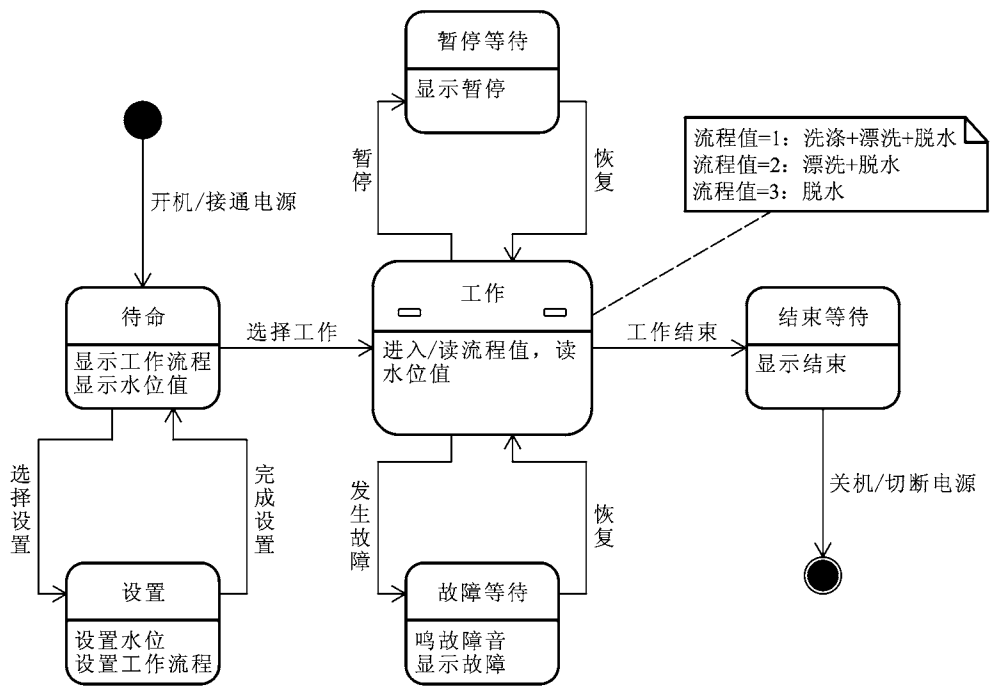


图 4-13 全自动洗衣机的状态图模型

需要注意的是，图 4-13 中的工作状态是一个复合状态。也就是说该状态中包含了子状态。假如洗衣机的常规工作流程是：累积洗涤 10 分钟，然后进入漂洗状态；累积漂洗 6 分钟，然后进入脱水状态；累积脱水 1 分钟，然后进入结束鸣音状态。则工作状态的下层子状态图如图 4-14 所示。

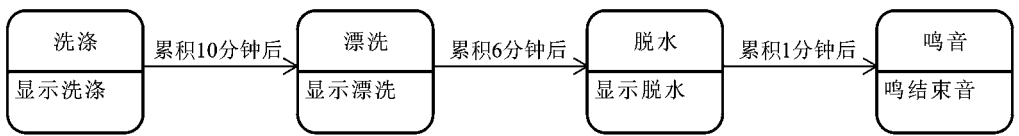


图 4-14 洗衣机工作状态的下层子状态图

4.5 需求有效性验证

需求有效性验证是指对已经产生的需求结论所要进行的检查与评价，它是需求分析过程中一个必不可少的环节。

需求分析是软件开发过程中一个非常关键的阶段。它是软件设计、实现的基础，同时也是用户对软件产品进行确认的基本依据。但是，需求分析过程中产生出来的结论难免存在问题。例如，某项功能被遗漏了，某些功能之间发生了操作上的冲突，某些数据字节数不够大等。更加重要的是，这些问题看起来或许并不显眼，但它所影响的是软件系统的整体构造。

实际情况往往是：需求分析中的小问题，假如是在后续的开发过程中或是在系统开发出来并投入使用以后才被发现，就会导致代价巨大的返工。因此，在需求分析结果出来以后，需要对其进行严格的有效性验证，由此尽早地发现需求文档草稿中存在的问题。

#### 4.5.1 需求验证内容

在需求有效性验证过程中，为了确保软件需求的正确性，需要对需求文档草稿从有效性、一致性、完整性、现实性等几个方面进行有效性验证。

##### 1. 有效性验证

需求有效性验证用于确认每一项需求定义都能符合用户的实际需要。由于一个软件系统在其运行过程中一般需要面对许多不同的用户，他们分别会有一些不同的个性需求意愿，因此，有效性验证还包括对用户需求个性差异的协商。

##### 2. 一致性验证

一致性验证用于检查发现在需求文档中可能存在的需求冲突，例如，同一个功能出现了不同的描述或存在相互矛盾的规程约束。

##### 3. 完整性验证

完整性验证用于检查发现用户确实需要的，但没有写入需求文档中一些功能、约束等，以使最终确定下来的需求文档能够更加完整的体现用户的需求意愿。

##### 4. 现实性验证

现实性验证用于检查需求文档中所提出的功能、性能、安全等方面的需求，哪些还不能够利用现有技术加以实现，以确保用户的一系列需求都能够具有现实意义，都能够最终实现。

##### 5. 可检验性验证

可检验性验证用于判断需求文档中的各项需求是否有适合于用户操作的检测方法，可以使得当系统开发完成并交付用户使用，开发人员能设计出一组合适的检查方法来进行用户需求检验，由此最大限度地保证用户的需求意愿能够得以实现。

#### 4.5.2 需求验证方法

在进行需求有效性验证时，需要有一定的方法、工具提供支持。例如，需求评审、需求原型评价和基于 CASE 工具的需求一致性分析等。

##### 1. 需求评审

需求评审是传统的需求检查手段，采用专门评审小组的方式实施对需求文档的有效性评价。其评审工作的开展需要有开发人员、用户的共同参与，他们一同检查文档中的不规范之处和遗漏之处，一起讨论需求中存在的问题，并需要对一些需求分歧进行协商。

需求评审可以是非正式的也可以是正式的。其中，非正式评审一般是在正式评审之前进行的准备性工作。非正式评审往往由项目负责人召集，由尽可能多的项目相关人员一起参与讨论，然后就诸多需求结论是否正确给出一个基本判断。

在非正式评审结果产生以后，接着需要进行正式的需求评审。这时，软件开发机构需要拿着经过非正式评审的需求文档访问用户，逐条地向各个用户解释需求含义。

在正式评审过程中，软件评审小组一般需要针对以下问题进行专门的检查与评价：

- (1) 一致性：需求文档对需求的描述是否有不一致的地方？
- (2) 完整性：是否还有需要的但没有写到需求文档中的功能？
- (3) 可检验性：需求描述是否可实际测试？
- (4) 可读性：需求描述能否被用户读懂？
- (5) 可跟踪性：是否清晰地记录了需求的出处？
- (6) 可调节性：需求是可调节的吗？假如需求出现变更，能否不对系统带来太大的影响？

## 2. 需求原型评价

本章 4.3.3 节主要说明了如何使用需求原型完善用户的需求，但在这个过程中也包括对用户需求的有效性验证。例如，可以根据需求文档为用户创建一个可以运行的系统模型，使用户通过模型进行更加接近实际的系统需求检验。

需求原型主要用来提供给用户评价，以方便用户进行需求确认。为了使需求原型评价更加有效，分析人员有必要从不同方面对用户评价作些引导。例如，为了启发用户的评价行为，可以针对界面原型提出以下问题：

- (1) 界面所显示的功能是你所期望的吗？
- (2) 有遗漏的功能吗？
- (3) 有多余的功能吗？
- (4) 你感觉界面复杂吗？

需求原型评价还有利于发现用户的一些潜在需求。因此，在通过原型进行需求验证的时候，除了需要从用户对原型的评价中确认用户的想法之外，还有必要观察用户对原型的操作，以此发现用户所需要的而未能表达出来的需求。

用于需求评价的原型一般是抛弃型原型，当需求被确定之后，原型会被抛弃。这种原型能够快速创建，容易修改，可以加快需求工程速度，降低需求成本。

需求原型也可以是进化型原型，特别是当开发者在需求原型上花费时间太多的时候，就自然会产生出要把原型进化为目标系统的想法。若要使需求原型能够进化则一般需要满足以下两个条件：

其一：原型创建工具和目标系统创建工具应该尽量一致，以方便从原型到目标系统的无缝过渡。例如，使用 Microsoft Visual Basic、Inprise Delphi 等基于组件的可视化开发工具作为原型创建工具。

其二：原型创建时必须考虑到软件的健壮性、可靠性、可维护性，以及工作性能等技术性要素，以保证原型质量标准 and 目标系统质量标准是一致的。

以上条件意味着，要使原型能够进化，开发者就要在原型创建时投入更多的时间、精力。显然这就不得不增大原型创建费用。一般来讲，项目越小则原型进化的价值也就越大。

## 3. 基于 CASE 工具的需求一致性分析

如果需求文档中的需求元素是用结构化或形式化语言描述的，则可以使用需求 CASE 工具来进行需求一致性分析。

需求 CASE 工具的工作流程如图 4-15 所示。其中，需求处理器用于处理使用形式化语言描述的需求，并将产生的需求结果存入需求数据库中。以后，需求分析器可以使用方法规则或符号规则对需求数据库中的需求结果进行一致性检查，并能够根据检查结论产生出关于需求一致

性的分析报告。

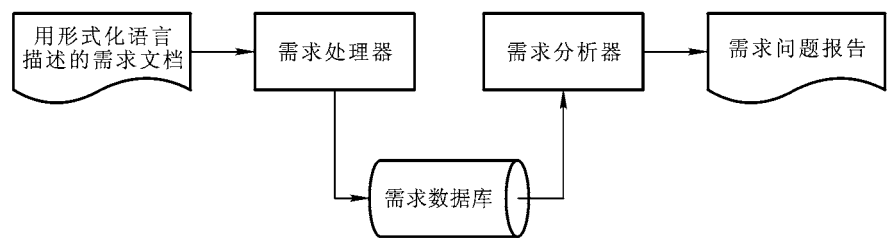


图 4-15 需求一致性分析工具

## 4.6 需求规格定义

需求规格说明书是需求分析阶段需要交付的基本文档，涉及引言、术语定义、用户需求、系统体系结构、系统需求等有关软件需求及其规格的诸多描述与定义。应该说，有关软件系统的一系列需求结论都需要以正式文档的形式写进这份文档之中。

需求规格说明书将成为开发者进行软件设计和用户进行软件验证的基本依据，其作用是使软件用户和软件开发者双方在软件正式开发之前能够对需要开发的软件有一个共同认可的规格定义。

需求规格说明书具有里程碑的意义，涉及比较广泛的读者群。几乎所有与软件项目有关的人员，包括用户、项目管理人员、系统开发人员、系统测试人员和系统维护人员等，都需要阅读这份文档，并或多或少地受到它的一定范围与程度的约束。

- 软件用户：提出需求，按照需求规格说明书对软件系统进行验收。
- 项目管理人员：根据需求规格说明书制定项目详细开发计划，安排项目进程。
- 系统开发人员：以需求规格说明书为依据进行系统设计与实现。
- 系统测试人员：以需求规格说明书为依据进行系统有效性测试。
- 系统维护人员：通过需求规格说明书来帮助对系统功能与构造的认识。

### 小 结

#### 1. 需求分析任务

##### (1) 用户需求

用户需求是用户关于软件的一系列意图、想法的集中体现，是用户关于软件的外界特征的规格表述。

##### (2) 系统需求

系统需求是比用户需求更具有技术特性的需求陈述，是提供给开发者或用户方技术人员阅读的，并将作为软件开发人员设计系统的起点与基本依据。主要包括：功能、数据、性能、安全等诸多方面的需求问题。

#### 2. 需求分析过程

需求分析是对软件系统的后期分析,需要进行的活动包括:分析用户需求、建立需求原型、分析系统需求和进行需求验证等。

### 3. 用户需求获取

(1) 用户调查是最基本的用户需求信息收集方法,比较常用的调查方法包括:访谈用户、开座谈会、问卷调查、跟班作业、收集用户资料。

(2) 需求原型可被用来解决用户对软件系统在需求认识上的不确定性。一般情况下,开发人员将软件系统中最能够被用户直接感受的那一部分东西构造成为原型。例如,界面、报表或数据查询结果。

### 4. 结构化分析建模

所谓模型,就是对问题所做的一种符号抽象。可以把模型看作为一种思维工具,利用这种工具可以把问题规范地表示出来。主要的分析模型包括:

(1) 功能层次模型。它使用矩形来表示系统中的子系统或功能模块,使用树形连线结构来表达系统所具有的功能层级关系。

(2) 数据流模型。用于描述系统对数据的加工过程,其图形符号是一些具有抽象意义的逻辑符号,主要的图形符号包括:数据接口、数据流、数据存储和数据处理。可以依靠数据流图来实现从用户需求到系统需求的过渡。结构化分析就是基于数据流的细化实现的,它是结构化分析方法的关键。

(3) 数据关系模型。也称为 ER 图,是应用最广泛的数据库建模工具。需要通过数据实体、数据关系和数据属性这三类图形元素建立数据关系模型。

(4) 系统状态模型。通过系统的外部事件、内部状态为基本元素来描绘系统的工作流程,这种建模方式比较适合于描述一些依赖于外部事件驱动的实时系统。

### 5. 需求有效性验证

需求有效性验证是指对已经产生的需求结论所要进行的检查与评价。

一般需要对需求文档草稿从有效性、一致性、完整性、现实性等几个方面进行有效性验证。比较常用的需求有效性验证方法与工具包括:需求评审、需求原型评价和基于 CASE 工具的需求一致性分析。

### 6. 需求规格定义

需求规格说明书是需求分析阶段需要交付的基本文档,将成为开发者进行软件设计和用户进行软件验证的基本依据,涉及引言、术语定义、用户需求、系统体系结构、系统需求等有关软件需求及其规格的诸多描述与定义。

## 习 题

1. 什么是用户需求?什么是系统需求?
2. 用户需求调查主要有哪些方法?
3. 说明需求分析过程中建立需求原型的作用。
4. 说明需求评审的作用。
5. 某“人力资源管理系统”的组成是:

(1) 人事管理子系统, 包括: 档案管理、岗位设置、人事异动等功能。

(2) 业绩管理子系统, 包括: 业绩标准设置、业绩录入、绩效评估等功能。

(3) 工资管理子系统, 包括: 工资标准设置、工资数据生成、工资结构分析、人力成本分析等功能。

(4) 用工管理子系统, 包括: 人力规划、人员调配、新人招聘等功能。

请使用功能层次图直观描述该“人力资源管理系统”的组织结构。

6. 某“零配件仓库管理系统”的系统流程图如图 4-16 所示。

(1) 假设该系统原有的各项功能都由“仓库管理员”操作, 请使用数据流图说明该系统的逻辑加工流程。

(2) 又假设准备对该系统进行改造。其中, 入库单将由采购部门输入, 出库单将由销售部门输入, 入出库分析报表将直接打印到计划部门, 采购定单将直接打印到采购部门, 而原来的“仓库管理员”仅承担入出库产品验证的任务。请说明改造以后的系统与原有系统的差别, 并使用数据流图说明改造后的该系统的逻辑加工流程。

7. 某银行储蓄系统的工作过程大致如下:

(1) 由储户填写存款单或取款单, 然后交由银行工作人员输入系统。

(2) 如果是存款, 系统将记录存款账号、存款人姓名、身份证号码、存款类型、存款日期、到期日期、利率等信息, 并会提示储户键入密码。在此之后, 系统会打印一张存款凭据给储户。

(3) 如果是取款, 则系统首先会根据存款账号核对储户密码。若密码正确, 则系统会计算利息并打印出利息清单给储户。

请使用数据流图分层描述该系统的逻辑加工流程。

8. 某图书馆图书管理系统对数据的基本要求如下:

(1) 涉及图书、图书管理员、读者这三类数据实体。

(2) 上述数据实体的数据结构是: 图书(图书编号、书名、作者、出版单位、出版日期、书价); 图书管理员(管理员编号、姓名、密码、登记日期); 读者(读者编号、姓名、工作单位、身份证号码)。

(3) 上述数据实体之间的关系是: 图书将由管理员登记入册, 在登记图书时需要记录图书登记日期; 图书将被读者借阅, 在图书办理借阅时需要记录图书的借书日期和还书日期。

请使用 ER 图建立上述数据的数据模型。

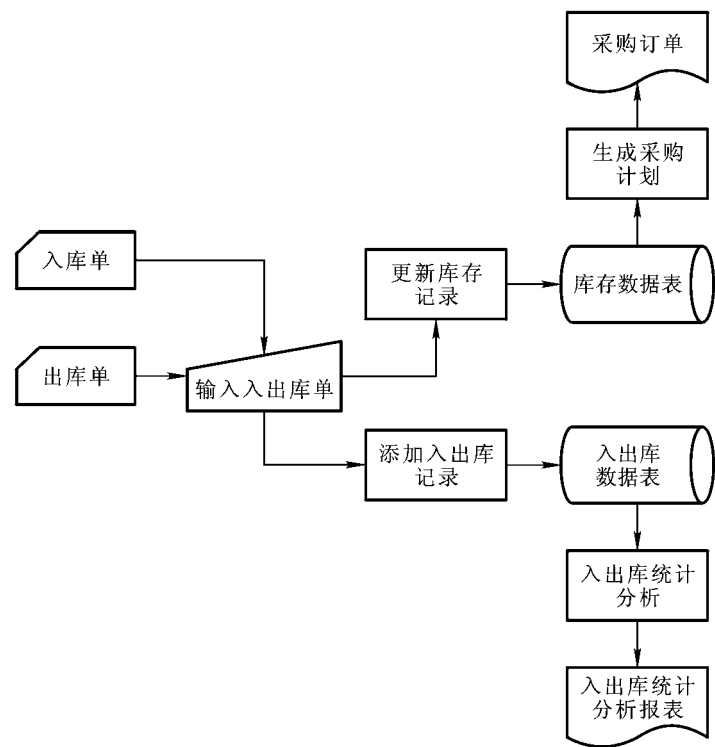


图 4-16 “零配件仓库管理系统”的系统流程图

9．某自动取款机系统工作过程大致如下：

(1) 在插入储蓄卡之前，自动取款机处于闲置状态。

(2) 在插入储蓄卡之后，自动取款机处于待命状态。

(3) 在储户输入密码之后，系统将对密码进行验证。若密码正确，自动取款机将进入工作状态；若密码不正确，自动取款机将提示储户出现输入密码错误。

(4) 在自动取款机进入工作状态以后，储户可选择“取款”或“退卡”。若选择“取款”，自动取款机将进入取款状态；若选择“退卡”，自动取款机将退出储蓄卡，然后进入闲置状态。

(5) 在自动取款机进入取款状态以后，储户可以输入取款金额，然后可选择“确定”或“取消”。若选择“确定”，自动取款机将进入付款状态，在完成付款之后，自动取款机返回到取款状态；若选择“取消”，自动取款机返回到取款状态。

请使用状态图描述该自动取款机的工作过程。



## 第5章 软件概要设计

在完成对软件系统的需求分析之后,接下来需要进行的是软件系统的概要设计。一般说来,对于较大规模的软件项目,软件设计往往被分成两个阶段进行。首先是前期概要设计,用于确定软件系统的基本框架;然后是在概要设计基础上的后期详细设计,用于确定软件系统的内部实现细节。

概要设计也称总体设计,其基本目标是能够针对软件需求分析中提出的一系列软件问题,概要地回答问题如何解决。例如,软件系统将采用什么样的体系构架、需要创建哪些功能模块、模块之间的关系如何、数据结构如何?软件系统需要什么样的网络环境提供支持、需要采用什么类型的后台数据库等。

应该说,软件概要设计是软件开发过程中一个非常重要的阶段。可以肯定,如果软件系统没有经过认真细致的概要设计,就直接考虑它的算法或直接编写源程序,这个系统的质量就很难保证。许多软件就是因为结构上的问题,使得它经常发生故障,而且很难维护。

### 5.1 概要设计过程与任务

#### 5.1.1 设计过程

概要设计基本过程如图 5-1 所示,它主要包括三个方面的设计。首先是系统构架设计,用于定义组成系统的子系统,以及对子系统的控制、子系统之间的通信和数据环境等;然后是软件结构和数据结构的设计,用于定义构造子系统的功能模块、模块接口、模块之间的调用与返回关系,以及数据结构、数据库结构等。

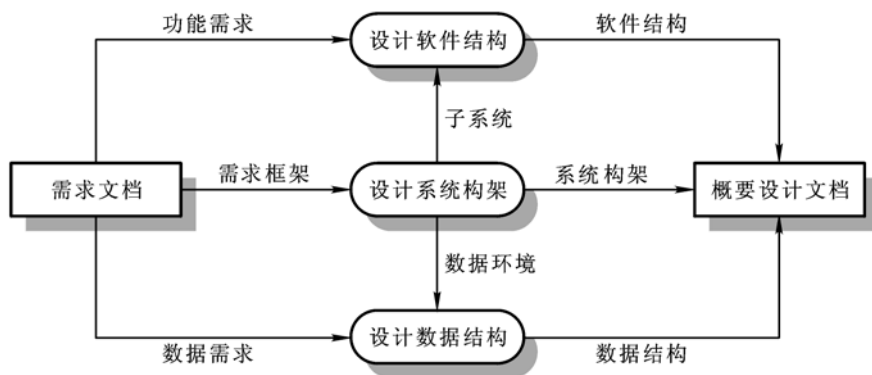


图 5-1 概要设计基本过程

概要设计要求建立在需求分析基础之上,软件需求文档是软件概要设计的前提条件。只有

这样，才能使得开发出来的软件系统最大限度地满足用户的应用需要。

实际上，概要设计的过程也就是将需求分析之中产生的功能模型、数据模型和行为模型等分析结论进行转换，由此产生设计结论的过程。在从分析向设计的转换过程中，概要设计能够产生出有关软件的系统构架、软件结构和数据结构等设计模型来。这些结论将被写进概要设计文档中，作为后期详细设计的基本依据，能够为后面的详细设计、程序编码提供技术定位。

需要注意的是，概要设计所能够获得的还只是有关软件系统的抽象表达式，需要专心考虑的是软件系统的基本结构，至于软件系统的内部实现细节如何，则被放到以后详细设计中去解决。例如模块，概要设计中的模块只是一个外壳，虽然它有确定的功能边界，并提供了通信的接口定义，但模块内部还基本上是空的，诸多具体的功能加工细节则必须等到详细设计完成以后才能确定下来。因此，在有关软件设计的全部工作中，概要设计所提供的并不是最终设计蓝图，而只是一份具有设计价值的具体实施方案与策略，用于把握系统的整体布局。尽管概要设计并不涉及系统内部实现细节，但它所产生的实施方案与策略将会最终影响软件实现的成功与否，并影响到今后软件系统维护的难易程度。

### 5.1.2 设计任务

概要设计阶段的任务既包括技术方面的，也包括管理方面的，具体说来，主要有以下几个方面：

#### 1. 制定规范

具有一定规模的软件项目总是需要通过团队形式实施开发，例如，组成一个或几个开发小组来承担对软件系统的开发任务。为了适应团队式开发的需要，在进入软件开发阶段之后，首先应该为软件开发团队制定在设计时应该共同遵守的规范，以便协调与规范团队内各成员的工作。

概要设计时需要制定的规范或标准包括：

- (1) 需要采用的管理规则，例如操作流程、交流方式、工作纪律等。
- (2) 设计文档的编制标准，包括文档体系、文档格式、图表样式等。
- (3) 信息编码形式，硬件、操作系统的接口规约，命名规则等。
- (4) 设计目标、设计原则。

#### 2. 系统构架设计

系统构架设计就是根据系统的需求框架，确定系统的基本结构，以获得有关系统创建的总体方案。其主要设计内容包括：

- (1) 根据系统业务需求，将系统分解成诸多具有独立任务的子系统。
- (2) 分析子系统之间的通信，确定子系统的外部接口。
- (3) 分析系统的应用特点、技术特点以及项目资金情况，确定系统的硬件环境、软件环境、网络环境和数据环境等。
- (4) 根据系统整体逻辑构造与应用需要，对系统进行整体物理部署与优化。

很显然，当系统构架被设计完成之后，软件项目就可按每个具有独立工作特征的子系统为单位进行任务分解了，由此可以将一个大的软件项目分解成许多小的软件子项目。

#### 3. 软件结构设计

软件结构设计是在系统构架确定以后，对组成系统的各个子系统的结构设计。例如，将子

系统进一步分解为诸多功能模块，并考虑如何通过这些模块来构造软件。

软件结构设计主要内容包括：

- (1) 确定构造子系统的模块元素。
- (2) 根据软件需求定义每个模块的功能。
- (3) 定义模块接口与设计模块接口数据结构。
- (4) 确定模块之间的调用与返回关系。
- (5) 评估软件结构质量，进行结构优化。

#### 4. 公共数据结构设计

概要设计中还需要确定那些将被许多模块共同使用的公共数据的构造。例如，公共变量、数据文件以及数据库中数据等，可以将这些数据看作为系统的公共数据环境。

对公共数据的设计包括：

- (1) 公共数据变量的数据结构与作用范围。
- (2) 输入、输出文件的结构。
- (3) 数据库中的表结构、视图结构以及数据完整性等。

#### 5. 安全性设计

系统安全性设计包括：操作权限管理设计、操作日志管理设计、文件与数据加密设计以及特定功能的操作校验设计等。概要设计需要对以上方面的问题作出专门的说明，并制定出相应的处理规则。

例如操作权限，假如应用系统需要具有权限分级管理的功能，则概要设计就必须对权限分级管理中所涉及的分级层数、权限范围、授权步骤以及用户账号存储方式等，从技术角度作出专门的安排。

#### 6. 故障处理设计

软件系统工作过程中难免出现故障，概要设计时需要对各种可能出现的来自于软件、硬件以及网络通信方面的故障作出专门考虑。例如，提供备用设备、设置出错处理模块、设置数据备份模块等。

#### 7. 可维护性设计

软件系统在投入使用以后必将面临维护，如改正软件错误、扩充软件功能等。对此，概要设计需要作出专门安排，以方便日后的维护。例如，在软件中设置用于系统检测维护的专用模块；预计今后需要进行功能扩充的模块，并对这些接口进行专门定义。

#### 8. 编写文档

概要设计阶段需要编写的文档包括：概要设计说明书、数据库设计说明书、用户操作手册。此外，还应该制定出有关测试的初步计划。

上述文档中，概要设计说明书是概要设计阶段必须产生的基本文档，涉及系统目标、系统构架、软件结构、数据结构、运行控制、出错处理、安全机制等诸多方面的设计说明。

#### 9. 概要设计评审

在诸多概要设计任务完成之后，应当组织对概要设计的评审。

概要设计评审内容主要包括：

- (1) 需求确认：确认所设计的软件是否已覆盖了所有已确定的软件需求。
- (2) 接口确认：确认该软件的内部接口与外部接口是否已经明确定义。
- (3) 模块确认：确认所设计的模块是否满足高内聚、低耦合的要求，模块的作用范围是否在其控制范围之内。
- (4) 风险性：该设计在现有技术条件下和预算范围内是否能按时实现。
- (5) 实用性：该设计对于需求的解决是否实用。
- (6) 可维护性：该设计是否考虑了今后的维护。
- (7) 质量：该设计是否表现出了良好的质量特征。

## 5.2 系统构架设计

大型的综合应用系统大都是由许多子系统组成的。一般说来，这些子系统能够独立运行，有自己专门的服务任务，并可能需要部署在不同的计算机上工作。

应该说，组成系统的子系统具有一定的独立性，但子系统之间又有着联系。例如，有共同的数据源，相互之间需要通信，并可能需要协同工作。系统构架设计的任务就是根据需求规格中的需求基本框架，把组成系统的这些子系统、子系统之间的关系、它们之间需要的数据通信等确定下来，并把它们工作时所需要的设备环境、网络环境和数据环境等也一同确定下来，由此对系统作出一个合理的、符合应用需要的整体部署。

需求分析中的需求框架是基于用户应用域建立的，概要设计时可以通过需求框架来映射系统构架。例如，可以利用需求分析中的高层数据流图对系统基本工作流程的描述，来映射系统的基本结构，使得需求分析中对系统的逻辑描述，转换为概要设计中对系统的物理描述。

一般情况下，系统构架设计可以按照以下步骤进行。

(1) 定义子系统。根据需求分析中有关系统的业务划分情况，将系统分解成诸多具有独立任务的子系统。

(2) 定义子系统外部接口。分析子系统之间的通信与协作，以获得对子系统外部接口的定义。

(3) 定义系统物理构架。根据系统的整体逻辑结构、技术特点、应用特点以及系统开发的资金投入情况等，选择合适的系统物理构架，包括：硬件设备、软件环境、网络结构和数据库结构，并将子系统按照所选的物理构架进行合理部署与优化。

下面将介绍几种典型的系统构架。需要注意的是，任何一种结构都会有优点与缺点，尽管是一些现在看来已经过时的结构也有它存在的现实价值。

### 5.2.1 集中式结构

集中式结构是最传统的系统构架，系统由一台计算机主机和多个终端设备组成，其结构如图 5-2 所示。

集中式结构的特点是系统中的全部软件资源都被集中安装在这一台主机上，包括：操作系统、数据库系统、应用系统和资源文件等。系统的智能处理器也被集中在主机上。用户则是通过和主机连接的基本无智能的终端设备与系统进行通信。由于所有计算任务都通过主机完成，

因此集中式结构对计算机性能有比较高的要求。早期应用中一般使用小型以上计算机提供比较强大的主机计算支持，操作系统则一般是 Unix 系统，能够向外提供多用户服务。

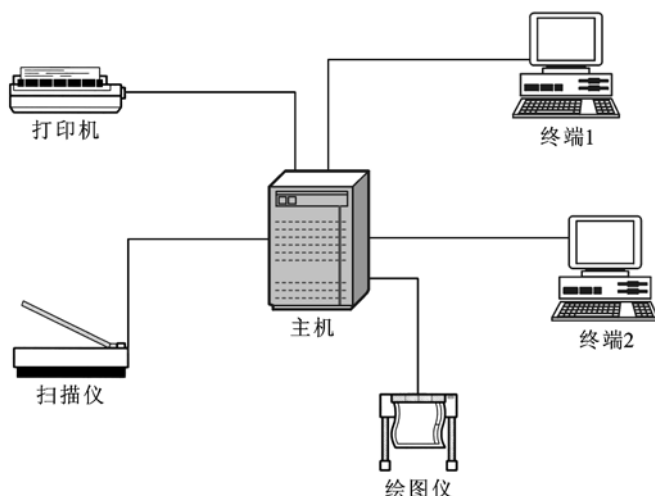


图 5-2 集中式结构

应该说，集中式结构具有非常好的工作稳定性。另外，由于集中式结构是直接通过终端接口实现多用户通信，因此系统还具有较高的安全保密特性。集中式结构优点是高稳定性和高安全性。但集中式结构有较苛刻的设备要求，系统建设费用、运行费用都比较高，而且系统灵活性不够好，系统结构不便于扩充。

### 5.2.2 客户机/服务器结构

客户机/服务器结构是一种分布与集中相互结合的结构，系统依靠网络被分布在许多台不同的计算机上，但通过其中的服务器计算机提供集中式服务。

图 5-3 所示的是客户机/服务器结构的典型应用。这是一个提供视频信息和图片信息的多用户超文本应用系统，系统中有多个服务器，能够分别提供不同的信息服务。其中，视频服务器提供视频数据服务，传送的数据需要快速同步，但分辨率相对较低；图片服务器提供图片数据服务，数据需要以高分辨率发送；目录服务器则提供目录查询服务，能够支持各种查询，并能够与超文本信息进行链接。

与集中式结构中的无智能终端不同，客户机/服务器结构中的客户机是具有智能的，需要安装客户程序，并需要通过客户程序访问服务器。例如图 5-3 中的诸多客户机，它们可以通过一个用户界面客户程序对服务器进行访问，并可以通过这个用户界面程序显示从服务器返回的图片或视频信息。

在客户机/服务器结构中，客户机是主动的，它主动地向服务器提出服务请求；而服务器则是被动的，它被动地接受来自客户机的请求。一般说来，客户机在向服务器提出服务请求之前，需要事先知道服务器的地址与服务；但服务器却不需要事先知道客户机的地址，而是根据客户机主动提供的地址向客户机提供相应的服务。

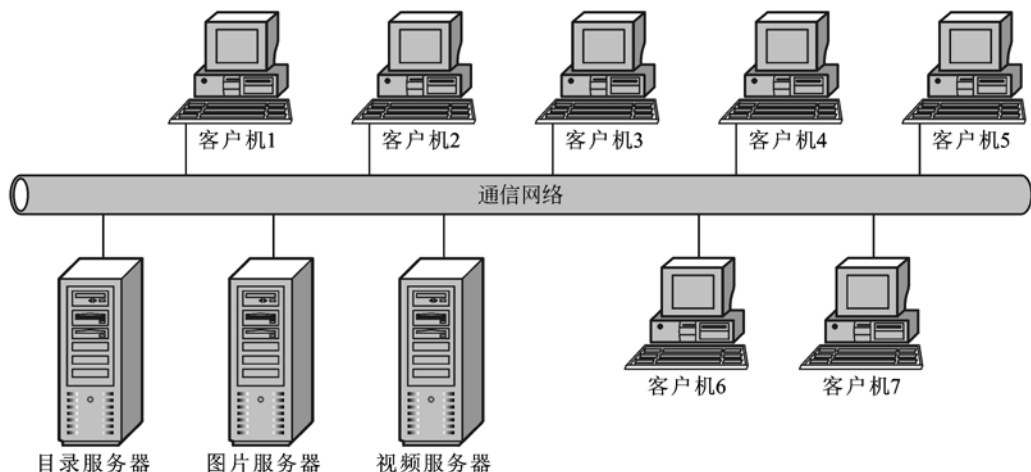


图 5-3 客户机/服务器结构

客户机/服务器结构的优越性是结构灵活、便于系统逐步扩充。例如上面的多用户超文本应用系统，也许用户最初只需要提供图片信息服务，因此系统初期开发时可以只创建图片服务器，只是当用户需要提供视频信息服务时，才创建视频服务器。显然，这有利于应用系统的分阶段实现与逐步扩充。另外，由于客户机/服务器中服务器可以采用高性能微机配置，因此其建设成本、运行费用等也明显低于基于小型机的集中式结构。

需要注意的是，客户机/服务器结构是一种逻辑结构，客户机或服务器都只是角色概念。因此可以将客户软件和服务软件安装在一台计算机上，而使这台计算机既担任客户机角色，又担任服务器角色。当然，也可以在一台计算机上安装多个服务软件，而使这台计算机承担多个服务器角色，例如，它可以既是图片服务器，又是视频服务器。

### 5.2.3 多层客户机/服务器结构

客户机/服务器结构已被广泛应用基于数据库的信息服务领域，并演变出了多层客户机/服务器结构。

图 5-4 所示是一个有关信息管理应用的逻辑结构。这是一个三层逻辑结构，其中的表示层是指对用户的信息服务；应用层是指对数据的应用逻辑加工；数据层是指系统中的数据库管理。在集中式系统中，以上各个不同层面的元素都被部署在一台主机上，这时或许没有必要在它们之间清楚地划分边界。但值得注意的是，当采用客户机/服务器结构时，这些元素却有可能要被分布到不同的计算机上去，因此必须在它们之间给出一个清楚的边界。

#### 1. 两层结构

两层客户机/服务器结构是将信息表示与应用逻辑处理都放在了客户机上，而服务器只需要管理数据库事务，其结构如图 5-5 所示。这时的客户机一般被称为“胖客户机”。

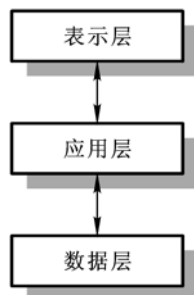


图 5-4 信息管理应用逻辑结构

两层结构的优点是结构简单，技术上比较容易实现。而且应用软件基本上是在客户端工作，因此非常方便客户端数据的计算与表现，能够有效保证系统的工作性能。

两层结构的缺陷是在管理与维护上存在困难。这时的客户端承担了信息表示与应用计算双重任务，客户端程序复杂，并且被分散在许多不同的客户计算机上。除非应用软件的功能非常稳定或客户端很少，否则当应用软件由于用户应用规则的变化，而不得不对其功能进行改造时，系统中数目庞大的客户机群，将会使系统变更显得非常麻烦，需要很大的系统变更开销。

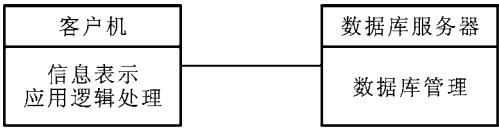


图 5-5 两层客户机/服务器结构

2. 三层结构

三层结构是鉴于两层客户机/服务器结构在管理与维护上存在的困难而专门提出的，这就是使“胖客户机”减肥，使它尽量简单，变成“瘦客户机”。更具体地说，就是将“胖客户机”中比较复杂，并且容易发生变化的应用逻辑部分提取出来，将它放到一个专门的“应用服务器”上，由此产生的结构如图 5-6 所示。

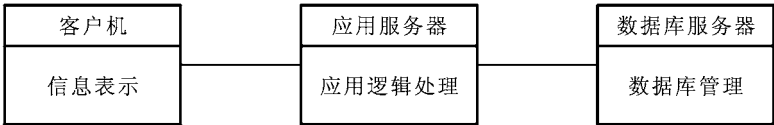


图 5-6 三层客户机/服务器结构

显然，三层结构使信息表示、应用逻辑处理和数据库管理被自然地分成了三个独立的部分。其中，应用逻辑处理是应用系统中的最易发生需求改变的部分，被放到了应用服务器上。与两层结构比较，三层结构给系统维护带来了很大的便利，当用户应用规则发生变化时，需要改变的不是数目庞大客户端，而是一个或少数几个应用服务器。但是，三层结构使实现软件的技术难度加大、开发周期延长，并且对服务器设备也有更高的要求。

需要指出的是，三层客户机/服务器结构也是逻辑结构，因此一个单一的服务器计算机可以既是应用服务器，又是数据库服务器。然而许多情况下，为了使系统具有更高的性能指标，或使系统具有更好的稳定性，系统中大都分别设置有专门的应用服务器和数据库服务器。而在一些实际开发中，为了使应用服务器能够承担来自于许多客户机的计算请求，还往往需要根据软件系统的计算负荷状态，对应用服务器进行专门的配置设计，例如，按事务进行任务分工，并在系统中设置多台专门应用服务器，它们分别完成一些特定的服务请求任务。

3. B / S 结构

B/S 结构是基于 Web 技术与客户机/服务器结构的结合而提出来的一种多层结构，其中的 B 是指 Web 浏览器（Browse），S 是指应用服务器与数据服务器（Server）。目前，这种结构已被广泛应用于网络商务系统之中，例如网上银行、网上商店等。其结构图如图 5-7 所示。

B/S 结构将信息表示集中到了专门的“Web 服务器”上。因此，与三层客户机/服务器结构比较，B/S 结构多了一层服务器。应该说，B/S 结构使客户端程序更加简化。这时的客户机上已经不需要专门的应用程序了，只要有一个通用的 Web 浏览器（例如，Microsoft Internet

Explorer)，就可以实现客户端数据的应用。

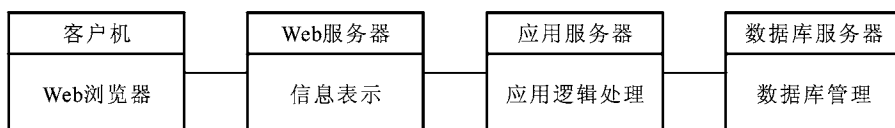


图 5-7 B/S 结构

B/S 结构的优点是不需要对客户机进行专门的维护（客户机上没有专门的应用程序），特别适合于客户机位置不固定或需要依靠互联网进行数据交换的应用系统。缺点是最终用户信息需要通过 Web 服务器获取，并通过网络传送到客户机上，因此系统的数据传输速度以及系统的稳定性，都将明显低于三层客户机/服务器结构。

如同其他客户机/服务器结构一样，B/S 结构也是逻辑结构，因此一个单一的服务器计算机可以既是 Web 服务器，又是应用服务器和数据库服务器。但如果需要使系统具有更高的性能或更加稳定的运行状态，那么则有必要将 Web 服务、应用处理和数据管理从物理上分离开来，设置专门的 Web 服务器计算机、应用服务器计算机和数据库服务器计算机。

在许多应用中，B/S 结构和三层客户机/服务器往往被结合起来使用。例如“网上购物系统”，其面向消费者的购物操作一般采用的是 B/S 结构，但面向购物中心工作人员的相关操作，为了保证系统运行稳定快捷，则可能会采用三层客户机/服务器结构。其对应的物理构架如图 5-8 所示。

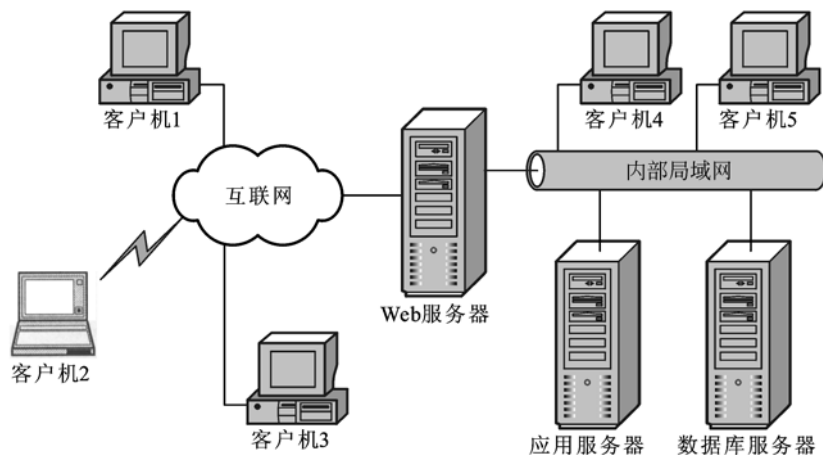


图 5-8 “网上购物系统”物理构架

#### 5.2.4 组件对象分布式结构

组件是一个对象包。组件对象分布式结构就是通过组件而使软件系统中的组件对象被分布到网络上的多台计算机上。组件对象具有一些公共接口，能够向外提供服务，不同组件的对象之间可以通过公共接口相互通信和协同工作。

应该说，传统的客户机/服务器结构是一种不对称的分布式结构，其客户机和服务器具有



不同的地位。客户机能够接受来自服务器的服务，但不能接受其他客户机的服务；服务器能够扮演客户机的角色，由此接受来自其他服务器的服务，但它不能请求来自客户机的服务。

然而，组件对象分布式结构则是一种比较对称的结构，在这种结构中，已经没有了客户机与服务器之间的界限，这些对象既可充当服务器，也可充当客户机，其角色只是决定于它是在提供服务还是在请求服务。

组件对象分布式结构是基于对象中间件建立的。主要用于对象分布式计算的中间件有：

- CORBA（通用对象请求代理模型）：由对象管理组织（OMG）定义，提供了一组通用的与机器无关的分布式对象计算方法。可用在 Unix、Linux 或 Windows 上。
- DCOM（分布式组件对象模型）：由微软公司定义，主要用在微软公司的操作系统上。

可以把对象中间件看成是能够在其上插拔组件的软件总线。这就像硬件总线允许不同的接口卡插于其上以支持硬件设备之间的通信一样，对象中间件作为软件总线，也允许在其上添加或移走组件对象，或进行组件对象之间的相互通信。其逻辑结构如图 5-9 所示。

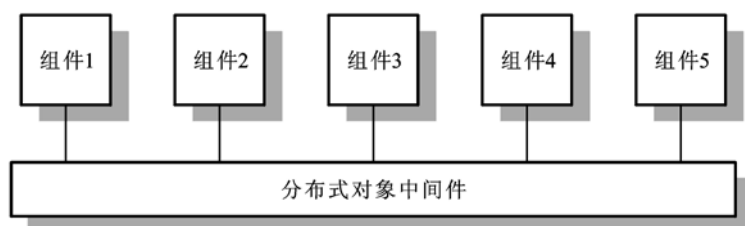


图 5-9 分布式组件对象逻辑结构

应该说，组件对象分布式结构是一个开放的体系构造，它不仅允许新的组件对象根据需要增加进来，而且还允许使用不同语言编写的对象。现在，组件对象分布式结构已经成为构造软件系统的一种基本模式，诸多运行在互联网上的远程服务系统，它们即是按照这种结构构造的。

在应用系统中采用组件对象分布式结构时，可依照以下步骤来进行系统结构设计：

- （1）按照系统功能服务对系统进行逻辑分块，确定系统的逻辑构造。
- （2）以系统逻辑构造为依据，设计系统的分布式组件对象。
- （3）以组件之间的通信与协作为依据，对组件进行物理部署。

组件对象分布式结构具有灵活的构架，系统伸缩性好，能够给系统的功能调整与扩充带来便利。但是，采用完全分布的组件对象结构，其技术难度太大，内部结构也较复杂。相比之下，客户机/服务器结构是一种更加容易理解的结构，并与现实的组织机构、业务过程具有一致性。因此，在许多实际应用中，往往将组件对象分布式结构与客户机/服务器结构结合起来，其中组件对象分布式结构被作为一种实现技术嵌入到客户机/服务器系统之中。这种构架的特点是：系统先按照客户机/服务器结构进行逻辑构造；然后无论是客户机还是服务器，都按照组件对象分布式结构进行内部构造设计，它们都由组件组成，并都需要通过对象中间件实现通信。显然，这样的系统将更加具有柔性，系统的变更也将会变得更加容易。例如，一个二层客户机/服务器结构，只需要对其组件对象进行重新部署，就可以将其改造成为三层甚至多层客户机/服务器结构。

实际上，分布式组件对象技术能够给应用服务器的创建带来更好的可伸缩性，由诸多组件

构造的应用服务器可以更加自由地进行物理部署。例如一个供销管理系统，假设其包括供应商联系、库存控制、客户联系、货物订购等业务内容，并且这些业务都由一个供销部门负责，则系统构造可按以上业务创建组件，并将这些组件部署在一台应用服务器上，如图 5-10 所示。

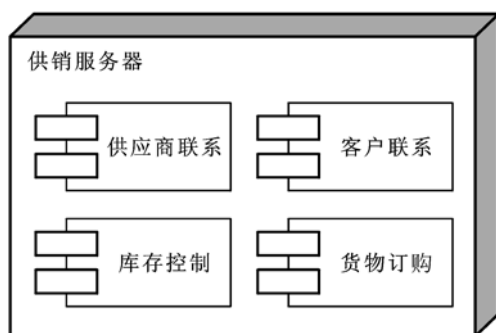


图 5-10 业务变更前的一台“供销”应用服务器

但假如业务流程发生了变化，例如，由于业务量扩大，该供销部门需要拆分为供应部、销售部两个部门时，考虑到业务归口与服务器负荷的影响，上面的一台应用服务器也需要改变为两台应用服务器，分别承担供应与销售的事务处理。显然，系统所采用的组件对象分布式结构能够很方便地适应这种业务需求的变更，需要进行的系统改造只不过是将用于业务处理的组件重新进行部署或安装，如图 5-11 所示。

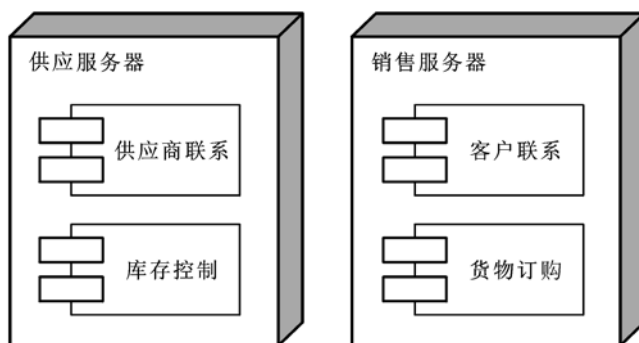


图 5-11 业务变更后的“供应”、“销售”两台应用服务器

## 5.3 软件结构设计

软件结构设计是对组成系统的各个子系统的进一步分解与规划。例如，将子系统按照其功能要素分解成具有一定的功能边界的模块，然后以模块为单位来构造软件。显然，需求分析阶段已经建立起的有关系统的功能模型、数据模型或状态机模型，可以作为软件结构设计的前提依据。

具体说来，软件结构设计包括以下几方面的内容：

(1) 确定构造子系统的模块元素。

- (2) 定义每个模块的功能。
- (3) 定义模块接口, 设计接口的数据结构。
- (4) 确定模块之间的调用与返回关系。
- (5) 评估软件结构质量, 进行结构优化。

### 5.3.1 模块概念

#### 1. 模块化

模块概念产生于结构化程序设计思想, 这时的模块被作为构造程序的基本单元, 例如函数、过程。

在结构化方法中, 模块是一个功能单位, 因此模块可大可小。它可以被理解为所建软件系统中的一个子程序系统, 也可以是子程序系统内一个涉及多项任务的功能程序块, 并可以是功能程序块内的一个程序单元, 例如函数、过程。也就是说, 模块实际上体现出了系统所具有的功能层次结构。

模块可以使软件系统按照其功能组成进行分解, 而通过对软件系统进行分解, 则可以使一些大的复杂的软件问题分解成诸多小的简单的软件问题。从软件开发的角度来看, 这必然有利于软件问题的有效解决。对于这个结论, 可以进行以下论证:

现假设函数  $C(P)$  用于度量问题  $P$  的复杂程度, 函数  $E(P)$  用于度量为解决问题  $P$  所需要的工作量 (用时间计算)。并假设有两个问题  $P_1$  和  $P_2$ 。

如果有:  $C(P_1) > C(P_2)$

则必有:  $E(P_1) > E(P_2)$

这是一个显而易见的经验性结论。它表明: 问题越复杂, 解决这个问题所需要的工作量就越大, 需要花费时间就越多。

另一个来源于经验的结论是:

$C(P_1 + P_2) > C(P_1) + C(P_2)$

它表明: 如果能够把两个结合在一起的问题隔离开来单独解决, 则可以使这原本结合在一起的问题的复杂程度下降。

显然, 从上面两个经验性结论中可以作出以下推论:

$E(P_1 + P_2) > E(P_1) + E(P_2)$

它表明: 如果能够把两个结合在一起的问题隔离开来单独解决, 则可以使这原本难于解决的问题变得容易解决起来。这个结论就是模块化的基本依据, 其作用就是能够使大的复杂的问题被“各个击破”。

上述结论表明: 模块化可以使软件问题简化。但是, 得出这个结论需要下面的前提条件: 大模块被分解成诸多小模块之后, 小模块基本上处于相互隔离的独立状态, 它们之间没有太多的通信。否则, 小模块之间由于存在太多的通信, 将会导致小模块接口复杂起来, 其结果是, 虽然每个模块内部简化了, 但整个软件问题反而复杂起来。

许多人都有一个良好的愿望, 那就是依靠模块化使系统不断分解而使整个系统不断简化, 由此使软件开发成本不断下降。然而这却可能做不到, 因为随着系统的分解, 系统中模块数目将会增加, 模块接口也会增加, 软件构造会由此变得复杂起来, 模块连接的难度也会由此加大。

实际上还有一个因素也在阻碍着这个愿望的实现，这就是软件系统的模块化分解本身也是一件并不轻松的工作，假如分解系统所需要的工作量，已经超过因为模块简化而减少的工作量，那就意味着，进一步分解系统已是一件得不偿失的事情了。

因此，软件系统模块化分解，从软件成本角度来看，有一个最小成本模块数范围。如图 5-12 所示，若分解程度低于这个最小成本模块数范围，则可继续分解，以降低软件开发成本；但若分解程度已经达到这个最小成本模块数范围，则就没有必要再进行分解了，否则将会反而增加软件开发的成本。

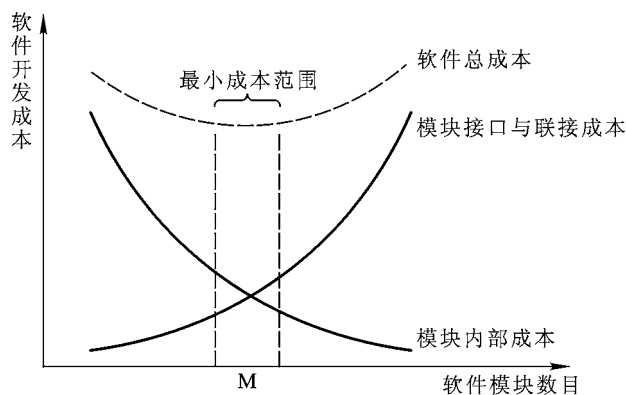


图 5-12 模块化与软件成本的关系

2. 抽象化

抽象是人所具有的一种高级思维活动，是以概括的方式抽取一系列事物的共同特征，由此把握事物的本质属性。抽象也是人类解决复杂问题时强有力的手段，能够使人从事物复杂的外部表象中发现事物的内在本质规律，由此找到解决问题的有效途径。

这种抽象的方法也被运用于软件工程之中。实际上，软件工程的每一个阶段，都能够看到抽象思维方法的作用，从软件的定义到软件的设计，直到软件编码与最终实现。但是，在软件工程的阶段，抽象层次与对象却各不相同，并需要运用不同的语言进行描述。例如，在软件分析阶段，抽象对象是系统的外部特征，需要采用的描述语言是软件问题所处环境中的用户术语；而在设计阶段，抽象对象则是系统的内部构造，需要采用的描述语言则是更加规范的适合于系统构造的过程化语言。

概要设计中的功能模块往往被看成是一个抽象化的功能黑盒子，其特征如图 5-13 所示。虽然它已是一个与软件实现直接相关的实体单元，可以看到它清晰的外观，但是却看不到它内部更加具体的实现细节。

抽象的作用是对事物现象的高度概括，但抽象的最终目的则是走到它的对立面，产生出具体的结果。因此，一个完整的抽象化过程是既包含抽象又包含具体的循环演变过程。应该说，正是由抽象到



图 5-13 概要设计中的模块

具体的不断演变，才使得人的抽象认识能够不断地产生出有意义的结果，并不断地推动着人类思想的进步。

这种由抽象到具体的不断演变也一直贯穿于软件工程过程之中，这就是自顶向下、逐步细化，其中，顶是抽象的，而细化则是这个抽象的顶具体化的结果。这意味着，软件工程的每一个阶段的推进，都具有从抽象到具体的转化。

例如需求分析，它建立在可行性研究基础之上。可行性研究所获得的是有关整个计算机系统的外部模型，这是一种对整个系统的高度抽象。但是，随着需求分析的进行，可行性研究中高度抽象的系统模型被逐步地转变成需求分析模型中每一个功能局部的规格定义。显然，这是软件从高层定义到低层定义的具体化演变。

又如概要设计，它是在需求分析基础上进行的，需要根据需求分析中的基本要求设计软件系统的基本构架，需要按照需求分析所提出的功能规格，确定软件系统中模块的构成，定义模块的接口，确定模块之间通信，设计对模块的控制。显然，这是软件从系统定义到系统设计的具体化演变。

应该说，概要设计中的结论，对于软件内部构造而言则仍是抽象的。比如概要设计中的功能模块，尽管它有惟一的名称标识，有明确的功能定义，并设置有数据的输入输出接口。但是，模块的内部实现细节则处于待定状态，需要等到详细设计完成以后才能得出更加具体的算法结论。

### 3. 信息隐蔽

设计软件结构时一个不可避免的问题是：为了得到一种高质量的模块组合，应该如何分解软件。对此，不得不了解什么是“信息隐蔽”。

信息隐蔽是指每个模块的内部实现细节对于其他模块来说是隐蔽的。也就是说，模块中所包含的信息，例如，模块内部的数据、语句或过程等，不允许其他不需要这些信息的模块使用。显然，信息隐蔽有利于模块相互之间的隔离，可以使每个模块更加具有独立性，并可以使模块之间的通信受到限制。例如，模块之间只能传递那些对于其功能实现而言是必须的信息。

通过限制模块需要使用的数据的作用范围，例如，定义模块内部局部变量，可以产生出模块内部信息隐蔽的效果。

模块内部信息隐蔽的好处是可以使软件系统更加健壮，更加方便维护。

以模块中的错误为例，假如模块内部信息是隐蔽的，则模块中存在的这个错误将比较难于扩散到其他模块。否则，系统可能因为一个小错误的扩散，而使整个系统崩溃。实际上，信息隐蔽还使软件错误定位更加方便。由于软件出错位置容易发现，因此，整个软件纠错工作的效率、质量都会随之提高。

一些模块的功能有时需要根据用户需求的变更而适时地进行一些功能改造。当需要对模块进行功能改造时，模块内部的信息隐蔽会使对模块改造所带来的影响限制在需要改造的模块之内，而与其他模块无关。显然，这将使软件系统的局部修改变得更加便利。

## 5.3.2 模块的独立性

模块的独立性是指软件系统中每个模块都只涉及自己特定的子功能，并且模块接口简单，与软件中其他模块没有过多的联系。

模块独立性是衡量软件中模块质量最重要的指标，是设计与优化软件结构时必须考虑的重要因素。当软件中的每个模块都具有很好的独立性时，软件系统不仅更加容易实现，并且会使今后的维护更加方便。

模块的独立性一般采用耦合和内聚这两个定性的技术指标进行度量。其中，耦合用来反映模块之间互相连接的紧密程度，模块之间的连接越紧密，联系越多，耦合性就越高。内聚用来反映模块内部各个元素彼此结合的紧密程度，一个模块内部各个元素之间结合越紧密，则它的内聚性就越高。显然，为了使模块具有较强的独立性，要求模块是高内聚、低耦合。

### 1. 耦合

耦合是软件结构中各个模块之间相互关联程度的度量。耦合的强弱取决于各个模块之间接口的复杂程度、接口数据对模块内部计算的影响程度和调用模块的方式。

模块之间的耦合形式主要有：非直接耦合、数据耦合、控制耦合、公共耦合和内容耦合。其中，非直接耦合和数据耦合是较弱的耦合，控制耦合和公共耦合是中等程度的耦合，内容耦合则是强耦合。

模块化设计的目标是尽量建立模块间耦合松散的系统。因此，在设计软件结构时一般也就要求尽量采用非直接耦合和数据耦合，少用或限制使用控制耦合和公共耦合，绝对不能使用内容耦合。

为了更好地认识模块之间的耦合，下面将对上述各种耦合形式给出必要说明。

#### (1) 非直接耦合

如果两个模块之间没有直接关系，它们之间的联系仅限于受到共同主模块的控制与调用，则称这种耦合为非直接耦合。

由于非直接耦合的模块之间没有数据通信，因此模块的独立性最强。

#### (2) 数据耦合

当一个模块访问另一个模块时，如果彼此之间是通过模块接口处的参数实现通信，并且参数所传递的数据仅用于计算，而不会影响传入参数模块的内部程序执行路径，则称这种耦合为数据耦合。

数据耦合是一种松散的耦合。依靠这种类型的耦合，模块之间既能实现通信，又有比较强的独立性。因此，软件结构设计中提倡使用这类耦合。

#### (3) 控制耦合

当一个模块访问另一个模块时，如果彼此之间是通过模块接口处的参数实现通信，并且参数传递了开关、标志、名字等控制信息，由此影响了传入参数模块的内部程序执行路径，则称这种耦合为控制耦合如图 5-14 所示。

由于接口参数影响着内部程序执行路径，因此控制耦合比数据耦合要强一些，会使模块的独立性有所下降。当需要通过一个单一的接

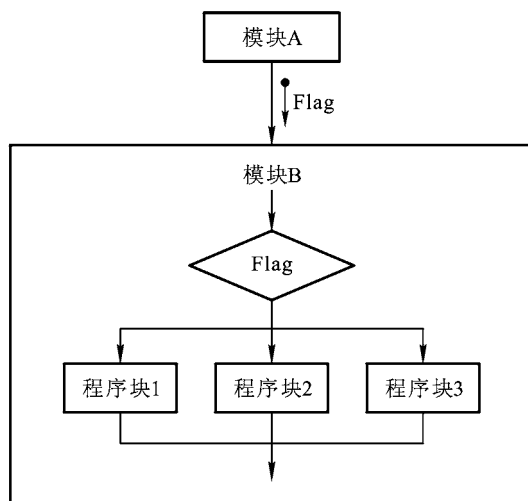


图 5-14 模块之间的控制耦合

口传递模块内多项功能的选择信息时，往往需要用到控制耦合。显然，在控制耦合形式下，对所控制模块的修改，需要受到控制参数的限制。

#### (4) 公共耦合

公共耦合是一种通过访问公共数据环境而实现通信的模块耦合形式，例如，独立于模块而存在的数据文件、数据表集、公共变量等，都可以看作为公共数据环境。

公共耦合中的公共数据环境是提供给任何模块的，当模块之间的耦合是公共耦合时，那些原本可以依靠接口提供的对数据的限制也就没有了。因此，相比依靠接口的耦合形式，公共耦合必然会使模块的独立性下降。也正因为如此，实际应用中，只有在模块之间需要共享数据，并且通过接口参数传递不方便时，才使用公共耦合。

需要注意的是：模块之间公共耦合的复杂程度，将会随着耦合模块个数的增加而显著增加。为了降低公共耦合带来的复杂性和提高模块的独立性，实际应用中还会针对公共耦合专门设置一些限制，例如不使用公共耦合数据传递控制信息。

图 5-15 中的模块采用了访问公共数据环境的公共耦合形式。其中图 (a) 是一个模块只往公共数据环境里送进数据，另一个模块只从公共数据环境中取出数据，这是一种比较松散的公共耦合；而图 (b) 则是两个模块都可以向公共数据环境里送进数据，又都可以从公共数据环境中取出数据，这就是一种比较紧密的公共耦合。

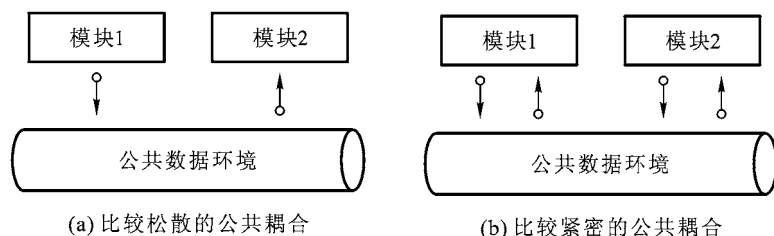


图 5-15 模块之间的公共耦合

另外，公共耦合还会带来以下一些方面的影响：

- 由于所有公共耦合模块都会与某一个公共数据环境有关，因此修改公共数据环境中某个数据的结构，将会影响到所有被耦合的模块。
- 由于没有办法控制各个模块对公共数据的存取，因此公共耦合会影响软件模块的可靠性和适应性。
- 由于公共数据环境需要被许多模块使用，因此不得不使用具有公共意义的数据名称。显然，这会使得程序的可读性有所下降。

#### (5) 内容耦合

如果发生下列情形，两个模块之间就发生了内容耦合。

- 一个模块直接访问另一个模块的内部数据。
- 一个模块不通过正常入口转到另一模块内部。
- 两个模块有一部分程序代码重叠。
- 一个模块有多个入口。

内容耦合是一种非常强的耦合形式，严重影响了模块独立性。当模块之间存在内容耦合时，

模块的任何改动都将变得非常困难，一旦程序有错则很难修正。因此，设计软件结构时，也就要求绝对不要出现内容耦合。所幸的是，大多数高级程序设计语言已经设计成不允许出现内容耦合，它一般只会出现在汇编语言程序中。

2. 内聚

内聚是对模块内部各个元素彼此结合的紧密程度的度量。模块内部各个元素之间的联系越紧密，则它的内聚程度就越高。模块的设计目标是尽量使模块的内聚程度高，以达到模块独立、功能集中的目的。

模块内聚的主要类型有：功能内聚、信息内聚、通信内聚、过程内聚、时间内聚、逻辑内聚和偶然内聚，如图 5-16 所示。其中，功能内聚和信息内聚属于高内聚，通信内聚和过程内聚属于中等程度的内聚，时间内聚、逻辑内聚和偶然内聚则属于低内聚；而且，模块内聚程度越高，其功能越集中、独立性越强。

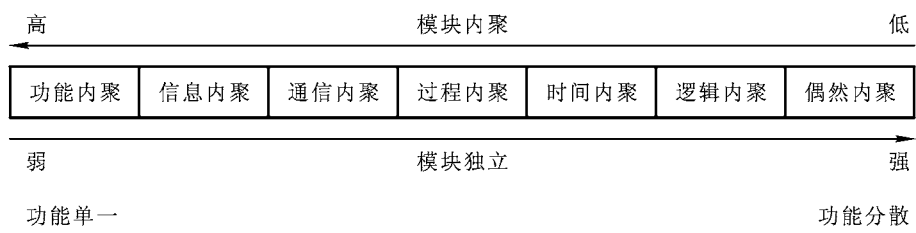


图 5-16 模块内聚类型比较

内聚所体现的是模块的内部功能构造，耦合所体现的是模块之间的联系。一般情况下，内聚和耦合是相互关联的，模块的内聚程度越高，则模块间的耦合程度就会越低，但这也不是绝对的。值得指出的是，比起模块之间的耦合来，模块的内聚更显重要。因此，实际设计中应当把更多的注意力放在如何提高模块的内聚程度上。

模块内聚的提高依赖于对模块功能的正确认识，应该通过定义使每一个模块都具有明确的功能。为了更好地认识模块内聚，下面将对上述几种内聚类型分别加以说明。

(1) 偶然内聚

当模块内各部分之间没有联系，或即使有联系，这种联系也很松散时，将会出现偶然内聚。

偶然内聚往往产生于对程序的错误认识或没有进行软件结构设计就直接编程。例如，一些编程人员可能会将一些没有实质联系，但在程序中重复多次出现的语句抽出来，组成一个新的模块，这样的模块就是偶然内聚模块。

偶然内聚模块由于是随意拼凑而成，模块内聚程度最低、功能模糊，很难进行维护。

(2) 逻辑内聚

逻辑内聚是把几种相关的功能组合在一起形成一个模块。在调用逻辑内聚模块时，可以由传送给模块的判定参数来确定该模块应执行哪一种功能。例如图 5-17 中的打印模块。

逻辑内聚模块比偶然内聚模块的内聚程度要高，因为它表明了各部分之间在功能上的相关关系。但是它每次执行的不是一种功能，而是若干功能中的一种，因此它不易修改。另外，在调用逻辑内聚模块时，需要进行控制参数的传递，由此增加了模块间的耦合。



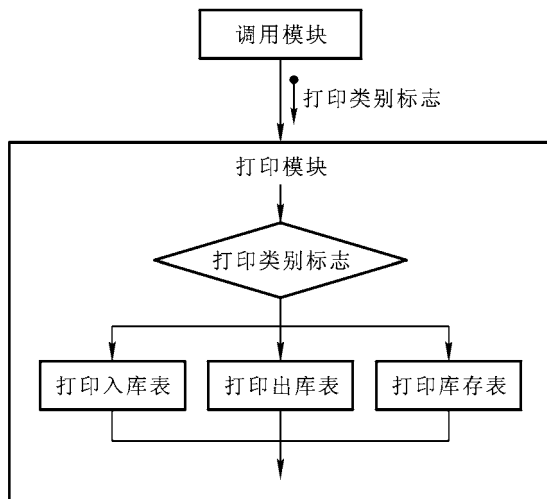


图 5-17 逻辑内聚

### (3) 时间内聚

时间内聚模块一般是多功能模块，其特点是模块中的各项功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行。例如初始化模块，其功能可能包括给变量赋初值、连接数据源、打开数据表、打开文件等，这些操作要求在程序开始执行的最初一段时间内全部完成。

时间内聚模块比逻辑内聚模块的内聚程度又稍高一些，其内部逻辑比较简单，一般不需要进行判定转移。

### (4) 过程内聚

如果一个模块内的处理是相关的，而且必须以特定次序执行，则称之为过程内聚模块。在使用流程图设计程序的时候，常常通过流程图来确定模块划分，由此得到的就往往是过程内聚模块。例如，可以根据流程图中的循环部分、判定部分和计算部分将程序分成三个模块，这三个模块就是过程内聚模块。

过程内聚模块的内聚程度比时间内聚模块的内聚程度更强一些，但过程内聚模块仅包括完整功能的一部分，因此模块之间的耦合程度比较高。

### (5) 通信内聚

如果一个模块内各功能部分都使用了相同的输入数据或产生了相同的输出数据，则称之为通信内聚模块。例如图 5-18 中的处理工资模块。

通信内聚模块由一些独立的功能组成，因此其内聚程度比过程内聚程度要高。

### (6) 顺序内聚

如果一个模块内的诸多功能元素都和某一个功能元素密切相关，而且这些功能元素必须顺序安排（前一项功能的数据输出作为后一项功能的数据输入），则称之为顺序内聚模块。当根据数据流图划分模块时，由此得到的通常就是顺序内聚模块。例如图 5-19 中的入库统计模块。

顺序内聚模块也包含着多项功能，但它有一个核心功能，其他功能则围绕着这个核心而安排。因此，顺序内聚程度比通信内聚程度更高。

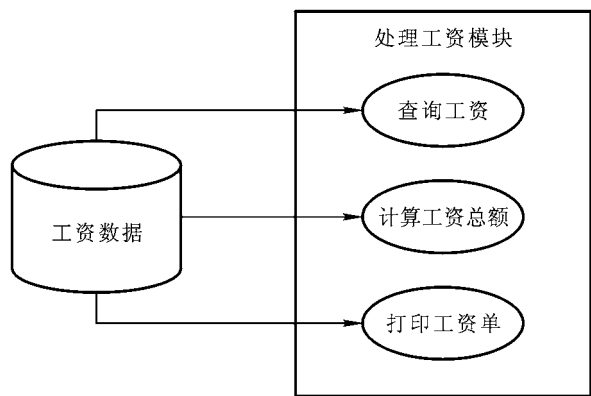


图 5-18 通信内聚

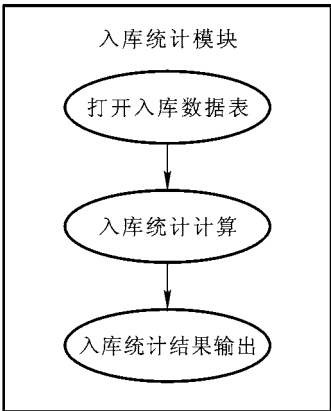


图 5-19 顺序内聚

(7) 功能内聚

如果一个模块中各个部分都是完成某一具体功能必不可少的组成部分，各个部分协同工作、紧密联系、不可分割，则称该模块为功能内聚模块。

功能内聚模块的特征是功能单一、接口简单，因此其容易实现、便于维护。与其他内聚类型相比，功能内聚具有最高的内聚程度，软件结构设计时应以其作为追求目标。

5.3.3 结构化设计建模

软件结构设计涉及模块功能、模块接口与模块调用关系等问题，为了使这些问题能够集中清晰地表达出来，软件结构设计需要借助于一定的图形工具来建立设计模型，例如软件结构图、HIPO 图。

1. 软件结构图

软件结构图由 Yourdon 于 20 世纪 70 年代提出，并被广泛应用于软件结构设计中，能够有效说明软件中模块之间的调用与通信。

软件结构图使用矩形框表示模块（框内注明模块的名字或主要功能），使用带箭头的直线段连接上下级模块，以表示上级模块对下级模块的调用。此外，软件结构图还可以在调用箭头旁使用带注释的箭头，以表示上级模块在调用下级模块时参数的传递与结果的返回，其基本图形符号如表 5-1 所列。

表 5-1 软件结构图的基本图形符号

图 形 符 号	说 明
	表示模块，框内注明模块的名称或主要功能
	表示调用，从上级模块指向下级模块
	表示传递或返回数据流
	表示传递或返回控制流

图 5-20 是一个自动阅卷系统的软件结构图。阅卷总控模块为顶层模块，其调用考卷数据输入、阅卷处理和考卷成绩输出这三个模块，以进行考卷输入、成绩计算和成绩输出的控制。在更下一级模块调用中，考卷数据输入模块通过调用读答卷卡模块将考卷原始数据输入，然后调用检验考卷数据模块产生出适合评分计算的有效数据；考卷成绩输出模块则通过调用格式化成绩数据模块对计算出的成绩数据进行输出前的格式化处理，然后调用写成绩记录模块实现成绩的存档操作。

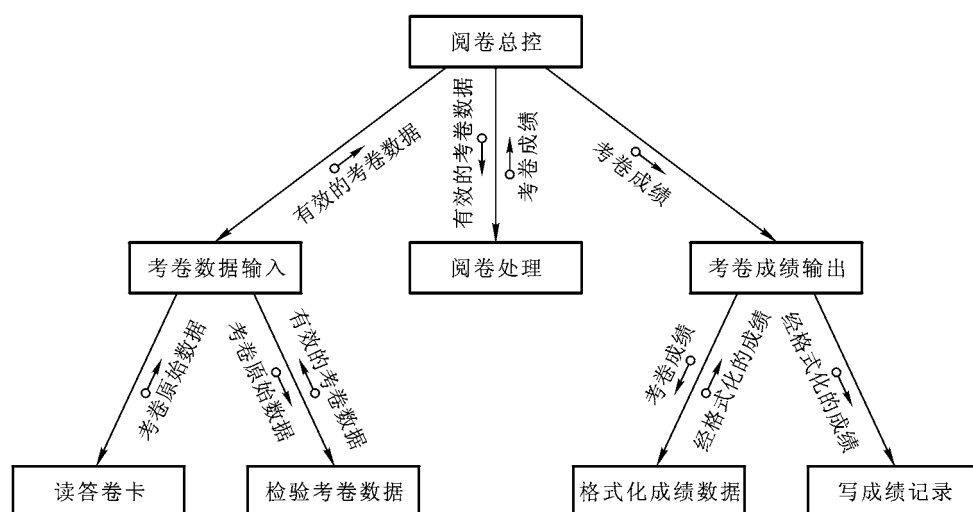


图 5-20 “自动阅卷系统”的软件结构图

软件结构图是一种画法比较灵活的软件结构设计工具，能够对复杂软件系统进行有效的描述，比较适合于设计人员在设计过程中对软件结构进行构思与设计说明。但软件结构图需要依靠带箭头的附加信息表示模块调用时的数据流和控制流，当软件结构图中包含了太多附加信息时，无形中会降低整个构图的清晰度。因此，软件结构图不太适合在正式文档中使用。

## 2. HIPO 图

HIPO 图是美国 IBM 公司推出的“H 图”与“IPO 图”的组合。

### (1) H 图

H 图是软件层次图的简称，用于描述软件结构上的分层调用关系，作用类似于软件结构图，但不涉及调用时的数据流、控制流等附加信息。

H 图的优点是清晰度高，能够用于正式文档中对软件结构的描述。图 5-21 是自动阅卷系统的 H 图，可以看出，它比上面的软件结构图要清晰得多。

为了能使 H 图中的模块具有可追踪性，由此可以与它所对应的 IPO 图联系起来。图中模块除了最顶层的之外，其他的模块都需要按照一定的规则编号，以方便检索。

### (2) IPO 图

IPO 图是“输入—处理—输出图”的简称，其中的“I”是指输入，“O”是指输出，“P”是指处理。可以使用 IPO 图对模块进行外部特征描述，涉及输入、输出接口和基本加工步骤等。在 HIPO 图中，需要针对 H 图中的每个模块给出 IPO 图描述。

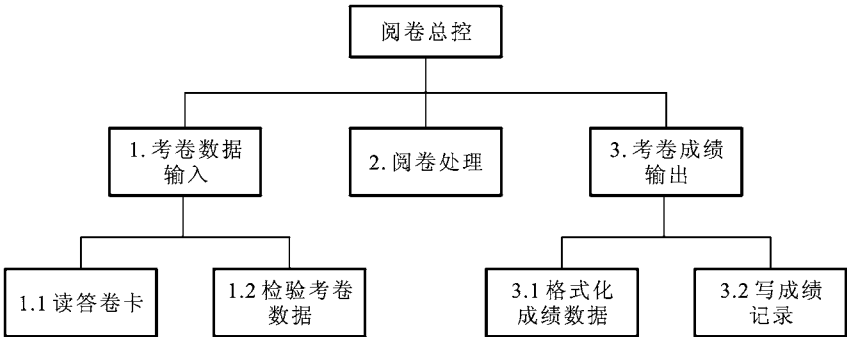


图 5-21 “自动阅卷系统”的 H 图

图 5-22 是图 5-21 中的“2. 阅卷处理”模块的 IPO 图。

系统名称：自动阅卷系统	模块名称：阅卷处理	模块编号：2
输入数据：有效的考卷数据		
输出数据：考卷成绩		
处理步骤： 1．循环累计 A 类题得分点 2．累计 A 类题得分数 3．循环累计 B 类题得分点 4．累计 B 类题得分数 5．考试成绩=A 类题得分数+B 类题得分数		

图 5-22 “阅卷处理”模块的 IPO 图

图 5-23 是图 5-21 中的“3. 考卷成绩输出”模块的 IPO 图。

系统名称：自动阅卷系统	模块名称：考卷成绩输出	模块编号：3
输入数据：考卷数据；经格式化的成绩		
输出数据：经格式化的成绩		
处理步骤： 1．调用“格式化成绩数据”模块，返回“有格式的成绩”数据 2．调用“写成绩记录”模块		

图 5-23 “阅卷成绩输出”模块的 IPO 图

从上面的 IPO 图举例可以看到，IPO 图提供了有关模块的更加完整的定义和说明。显然，这有利于由概要设计到详细设计的过渡。

### 5.3.4 软件结构优化

软件结构设计往往需要经历多次反复,其作用是可以不断地改进软件结构,提高软件质量。为了改进软件结构,软件设计人员进行了长期的实践与总结,由此得到了一些优化设计的原则。本节将介绍这些原则。

#### 1. 使模块功能完整

一个完整的功能模块,不仅能够完成指定的功能,而且还应当能够将完成任务的状态通知使用者。

具体说来,一个完整的功能模块应当包含以下几个部分的内容:

(1) 执行规定功能的部分。

(2) 出错处理的部分。当模块不能完成规定的功能时,必须回送出错标志,并向它的调用者报告出现这种例外情况的原因。

(3) 如果需要返回一系列数据给它的调用者,在完成数据加工时,应当给它的调用者返回一个该模块执行是否正确结束的“标志”。

#### 2. 使模块大小适中

许多情况下,可以用模块中所含语句的数量的多少来衡量模块的大小。有人认为限制模块的大小也是减少模块复杂性的有效手段之一,因而要求把模块的大小限制在一定的范围之内,例如将模块内的语句限制在 50~100 行左右。当然,这只能作为参考。

一般说来,模块过大的原因往往是模块的功能太多,因此有必要对大模块做进一步的分解。大多数情况下,可以从大模块中分解出一些下级模块或同层模块。

软件结构设计时,也有可能出现模块过小的问题。如果模块太小,则要注意这个模块的功能是否完整,是否将一个完整的功能分解到多个模块中去了。许多情况下,可以考虑将较小的模块与调用它的上级模块合并。如果模块虽小但功能内聚性好,或者它为多个模块所共享,或者调用它的上级模块很复杂,则不要贸然将小模块与其他模块合并,而是需要做更加细致的分析。

#### 3. 使模块功能可预测

一个功能可预测的模块可以被看成是一个“黑箱”,无论内部处理细节如何,只要输入数据是相同的,就总能产生相同的输出结果。

例如,在模块中使用了静态变量。因为静态变量会在模块内部产生较难预见的存储,而如果这个静态变量又被用为多项功能的选择标记,则可能会使得模块的功能难以被调用者预知。由于这个原因,设计者要特别慎重地使用静态变量。

模块功能可预测还意味着,模块的功能必须明确清晰地定义,应该使模块具有很好的内聚性。

#### 4. 尽量降低模块接口的复杂程度

模块接口是模块与外界进行通信的通道,较复杂的接口往往会带来较高的耦合。因此,应努力降低模块接口的复杂程度。对此,可以从以下两个方面作出考虑。

(1) 接口参数尽量采用简单数据类型,以使接口容易理解。例如,尽量使用基本字符、整数类型,而不是数组、指针、结合体类型。

(2) 限制接口参数的个数。接口参数个数太多,往往是由于模块功能混杂,不得不依靠参数进行调控。因此,过多的参数往往意味着模块还有进一步分解的必要。

#### 5. 使模块作用范围限制在其控制范围之内

模块的控制范围包括它本身及其所有从属于它的直接或间接下级模块。如图 5-24 所示,模块 B 的控制范围是模块 E、F、G、K,模块 C 的控制范围是模块 H、I、L、K,模块 D 的控制范围是模块 I、J、L。

模块的作用范围则是指模块内一个判定的影响范围,凡是受这个判定影响的所有模块都属于这个判定的作用范围。其中,如果一个判定的作用范围包含在这个判定所在模块的控制范围之内,则这种结构是简单的;否则,其结构就是不简单的,需要进行结构调整。

例如,图 5-24 中的模块 F。如果它做出一个判定之后,接着需要模块 G 工作,由于模块 G 不在模块 F 控制范围之内,因此模块 F 必须返回一个信号给模块 B,再由 B 把信号传送给模块 G。显然这不是一个好的设计,它增加了模块之间数据的传送量,并使模块之间出现了控制耦合,因此需要对其结构进行调整。

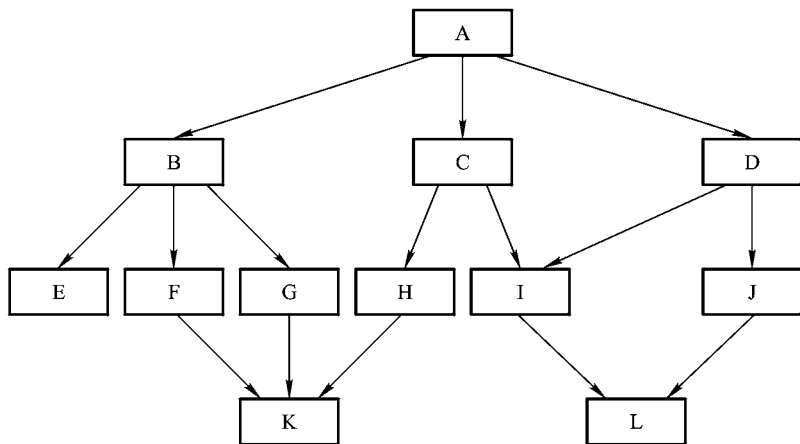


图 5-24 模块的控制范围与作用范围

在设计过程中,如果发现模块的作用范围不在控制范围之内时,可以采用如下办法进行结构调整,把模块的作用范围移到其控制范围之内。

(1) 将判定所在的模块合并到它的父模块中去,或上移到层次较高的位置上去,由此可使判定处于一个较高的位置,以达到有效的控制。例如,将模块 F 与模块 B 合并,如图 5-25 所示,由此可使模块 G 受到控制。

(2) 将受判定影响的模块下移到控制范围以内。例如,将模块 G 下移到模块 F 之下,如图 5-26 所示,由此可使模块 G 受到模块 F 控制。

需要注意的是:在改进模块的结构时,应当根据具体情况通盘考虑,既要尽量小范围地调整结构,使改进后的软件结构能够最好地体现问题的原来结构,又要考虑改进后的结构在实现上的可行性。

#### 6. 深度、宽度、扇出和扇入应当适当

软件的深度是指软件结构的层数。例如,图 5-24 中的软件结构,其深度为 4 层。

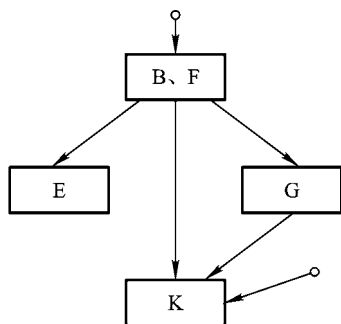


图 5-25 将模块合并到它的父模块中去

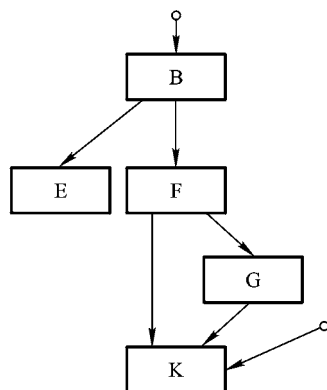


图 5-26 受判定影响的模块下移到控制范围以内

软件的宽度是指软件结构同一个层次上模块的总个数的最大值。例如，图 5-24 中的软件结构，其第二层宽度是 3，第三层宽度是 6，第四层宽度是 2。其整个软件的宽度是 6。

软件结构上的深度、宽度，在一定程度上反映出了软件系统的规模和复杂程度。也就是说，软件规模越大越复杂，其深度、宽度也会越大。一般说来，深度、宽度会受模块的扇出、扇入影响，并应该有一个合适的大小。深度、宽度太小，往往表示模块分解得还不够细，需要进一步分解。而深度、宽度太大，则可能表示模块分解得太细小了，或功能冗余模块太多了，以致软件结构变得复杂起来。一般说来，软件结构越复杂，模块之间的耦合就会越大。因此，假如软件结构过于复杂，就有必要将一些模块进行适当的合并，将那些功能相同或相似的模块合并起来作为公共模块使用。

模块的扇出是指模块直接调用的下级模块的个数。比较适当的扇出数为 2~5，一般不要超过 9。如果一个模块的扇出过大，就表明该模块具有过分复杂的控制功能，需要协调和控制过多的下属模块。对此，应当适当增加中间层次的控制模块，将比较集中的控制分解开来，如图 5-27 所示。但扇出过小也不好，这样将使得软件结构的深度大大增加，会带来过多的调用和返回的时间开销，由此降低软件的工作效率。

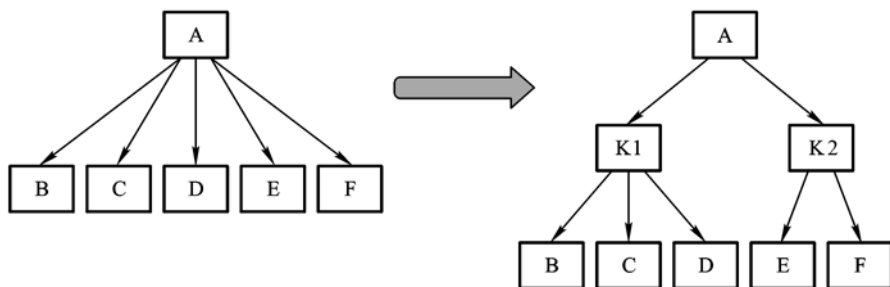


图 5-27 通过增加中间控制层可降低扇出数

模块的扇入是指模块受到了多少个直接上级模块的调用。一个模块的扇入越大，则共享该模块的上级模块的数目就越多。如果一个模块的扇入数太大，而它又不是公用模块，则说明该模块可能具有多项功能。在这种情况下，应当对它做进一步的分析，并将其功能做进一步的分解。

一般说来，各个不同层次的模块具有以下特点：顶层模块起全局控制作用；中间层次的模块起局部控制作用，并适当承担一些简单的操作提示功能；底层模块担任具体加工任务。因此，一个设计得比较好的软件结构通常应具有这样的特征：顶层模块高扇出，中层模块低扇出，底层模块高扇入。

## 5.4 面向数据流的结构设计

作为构造软件的基本框架，软件结构应该与需要分析时建立的分析模型保持一致。一种非常有效的设计思路是，基于需求分析中的数据流模型进行软件结构映射，由此产生出软件系统的基本设计模型。

为了方便从数据流模型中映射出软件结构来，需要对数据流进行合理的分类。例如，将数据流分为变换流或事务流，然后按照它们各自不同的特点分别采取不同的映射方法。

### 5.4.1 变换流分析与设计

#### 1. 变换流

变换数据流所体现出的是数据从输入到加工再到输出的一般步骤。变换数据流对数据的加工程序如图 5-28 所示，这就是数据首先需要经过输入过程，将外部数据形态转变为适合进行加工处理的内部形态；然后经过变换中心，将已转成内部形态的输入数据加工成一种新的数据形态；接着再经过输出过程，将经过加工产生的新的数据结果转换成适合向外导出的数据形态。

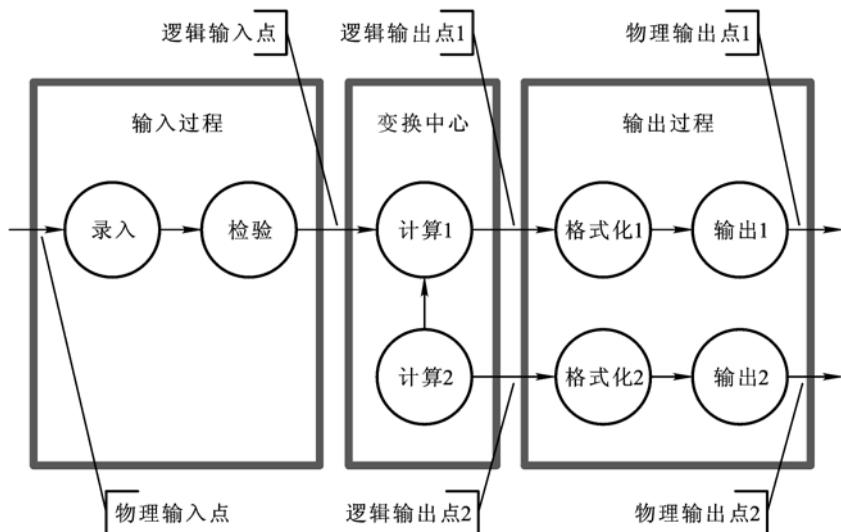


图 5-28 变换数据流

#### 2. 高层框架

由于变换数据流将整个过程分割成了输入、变换和输出三个部分，因此，对软件结构的映射也就可以是在总控模块之下，将软件分为输入、变换、输出三个部分。为了减轻总控模块的控制负担，可以针对这三个部分，分别设置控制模块，由此获得对各个区段的有效控制。



图 5-29 是基于变换流的高层框架图。

3. 下层模块

当软件结构的高层框架被确定下来以后，接着需要考虑的是那些涉及具体操作的下级模块。为了确定下级模块，并能够按照一定规则将这些下级模块挂接到上面的框架上去，有必要对变换数据流进行分段研究。

(1) 输入部分

一般情况下，可以把来源于外部端口的数据流称为物理输入点，而把由输入过程进入到变换中心的数据流称为逻辑输入点。图 5-28 中的输入部分从“物理输入点”到“逻辑输入点”，其模块挂接规则是从“逻辑输入点”往“物理输入点”进行搜索，所遇到的每一个处理框可以作为一个功能模块依次挂接到“输入控制”模块之下。

(2) 变换部分

图 5-28 中的变换中心涉及“计算 1”与“计算 2”两个处理框。可以将这两个处理框对应为功能模块直接挂接到“变换控制”模块之下。

(3) 输出部分

一般情况下，可以把由变换中心流向输出过程的数据流称为逻辑输出点，而把由输出过程流向外部端口的数据流称为物理输出点。图 5-28 中的输出过程从“逻辑输出点”到“物理输出点”，其模块挂接规则是从“逻辑输出点”往“物理输出点”搜索，所遇到的每一个处理框可以当做一个模块依次分层挂接。

依照上述方法，可以获得对图 5-28 中的变换数据流的软件结构映射，产生的软件结构如图 5-30 所示。

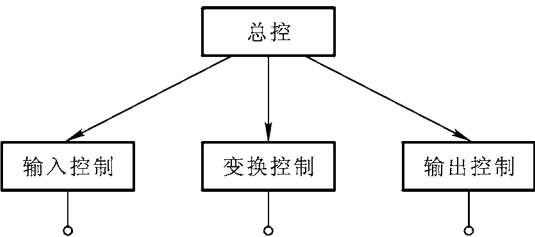


图 5-29 基于变换流的高层框架

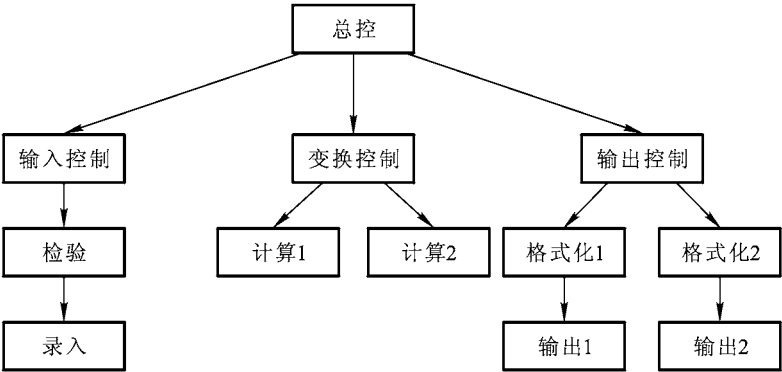


图 5-30 基于变换流的软件结构映射

5.4.2 事务流分析与设计

1. 事务流

当输入的数据流可以引发多个不同的事务活动流程，并且数据流图中有一个明显的事务调

度中心时,这种数据流被称为事务数据流。其特征如图 5-31 所示。需要注意事务流中的事务调度中心与变换流中的变换中心的区别,事务调度中心并不对输入数据进行加工,而只是根据不同的输入数据作出不同的事务流程选择。

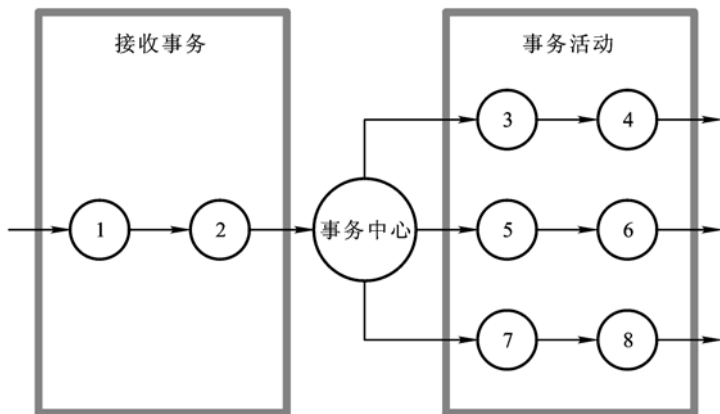


图 5-31 事务数据流

## 2. 软件结构

事务数据流以事务调度中心为核心,在此之前为接收事务,在此之后为事务分流处理。因此,基于事务流的软件结构映射也就可以是在总控模块之下将软件分为接收事务与事务活动两个部分。为了减轻总控模块的控制负担,可以针对这两个部分分别设置控制模块,例如“接收事务控制”模块、“调度事务控制”模块,以获得对各个区段的有效控制。

在事务流软件框架确定以后,接着需要考虑其下层模块的挂接。对此,以“事务调度中心”为起点,分别搜索事务输入流与事务活动流,将所遇到的每一个处理框当做一个功能模块依次挂接到“接收事务”模块或“调度事务”模块之下。

依照上述方法,可以获得对图 5-31 中的事务数据流的软件结构映射,产生的软件结构如图 5-32 所示。

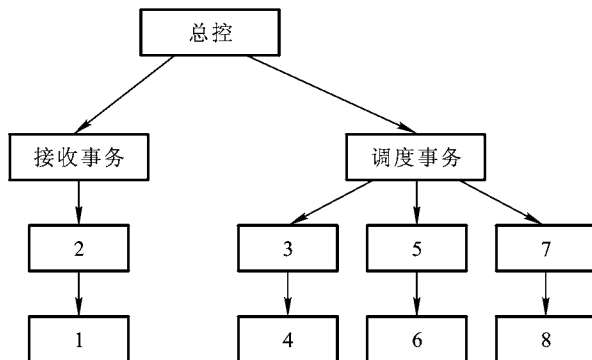


图 5-32 基于事务流的软件结构映射

### 5.4.3 混合流分析与设计

前面分别讨论了变换分析与事务分析。其中，变换分析是软件结构设计的主要方法，大部分软件系统都可以按照变换分析方法进行设计。但是，在很多情况下仅使用变换分析是不够的，还需要采用其他方法，事务分析就是一种非常有效的方法。例如商业数据处理系统，其主要组成部分就往往使用事务处理方法进行设计。

软件系统也可以是变换流与事务流的混合，如图 5-33 所示为典型的变换流与事务流的混合。对于这样的系统，通常采用变换分析为主、事务分析为辅的方式进行软件结构设计。其一般设计思路如下：

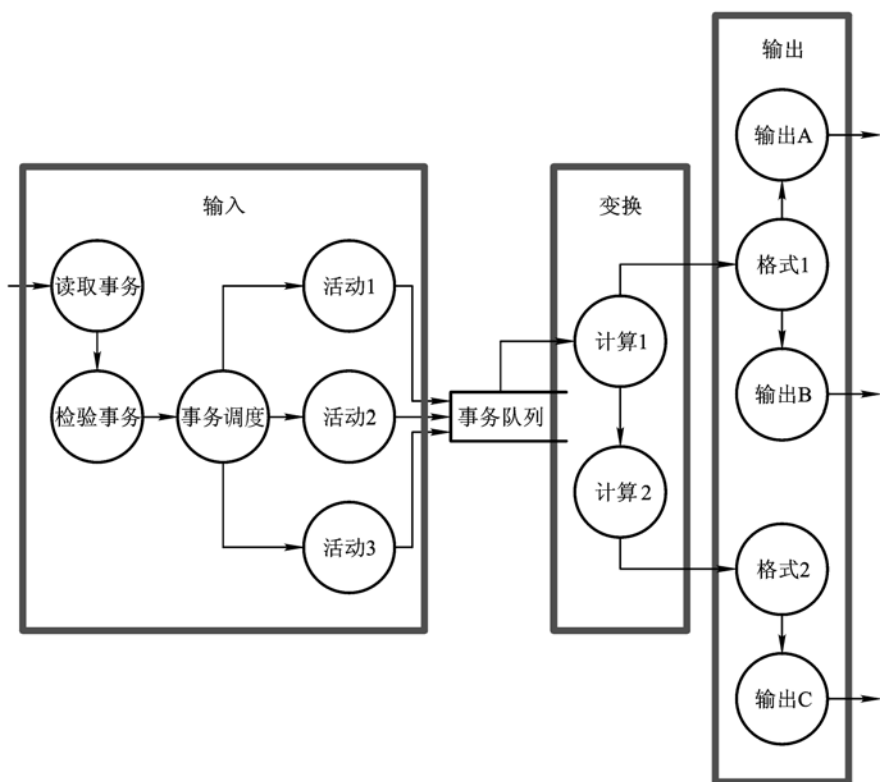


图 5-33 变换-事务混合型数据流

(1) 首先利用变换分析方法把软件系统分为输入、变换和输出三个部分，由此设计出软件系统的上层构架，例如顶层和第一层模块。

(2) 然后根据数据流图各部分的结构特点，适当地选择变换分析或事务分析，由此设计出软件系统的下层构架。

如图 5-33 中的混合数据流，即可以根据上述设计思路进行软件结构映射，由此可以产生出如图 5-34 所示的软件结构初始方案。

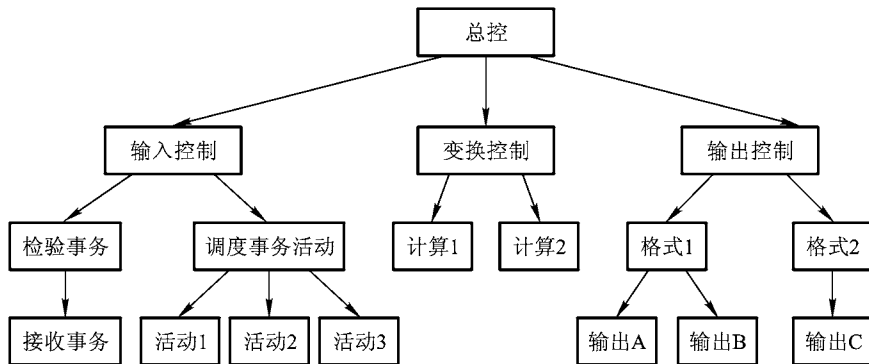


图 5-34 由混合型数据流映射的软件结构

#### 5.4.4 设计举例

问题：“邮件检测与计价电子仪器系统”软件结构设计。

该问题数据流如图 5-35 所示。这是一个典型的变换流问题，图中数据流已被划分为输入、变换和输出三个部分，并具有以下特征：

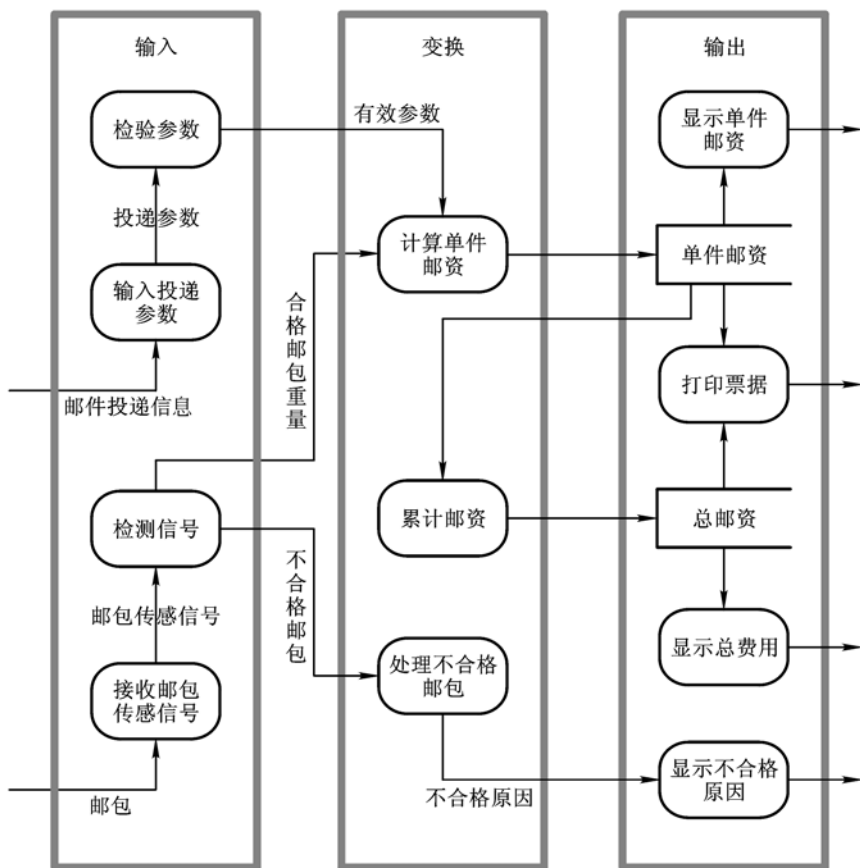


图 5-35 “邮件检测与计价电子仪器系统”数据流程图

### (1) 输入部分

有两个输入支路。支路一：用于输入与检验邮件投递参数，包括邮件种类、投递地区、邮资计价标准等参数；支路二：用于接收邮包传感信号并检测邮包，包括邮包重量、邮包封装状态、邮包违禁状态等信号。

### (2) 变换部分

包括计算单件邮资、累计邮资、处理不合格邮包等变换处理，分别用于合格邮包的邮资计价和不合格邮包的处理提示。

### (3) 输出部分

有四个输出支路，包括：显示单件邮资、显示邮资总费用、打印票据、显示不合格邮包处理提示。

基于上述对数据流的分析，可以方便地根据变换流的结构映射规则产生出它所对应的软件基本结构。

该问题软件结构如图 5-36 所示。

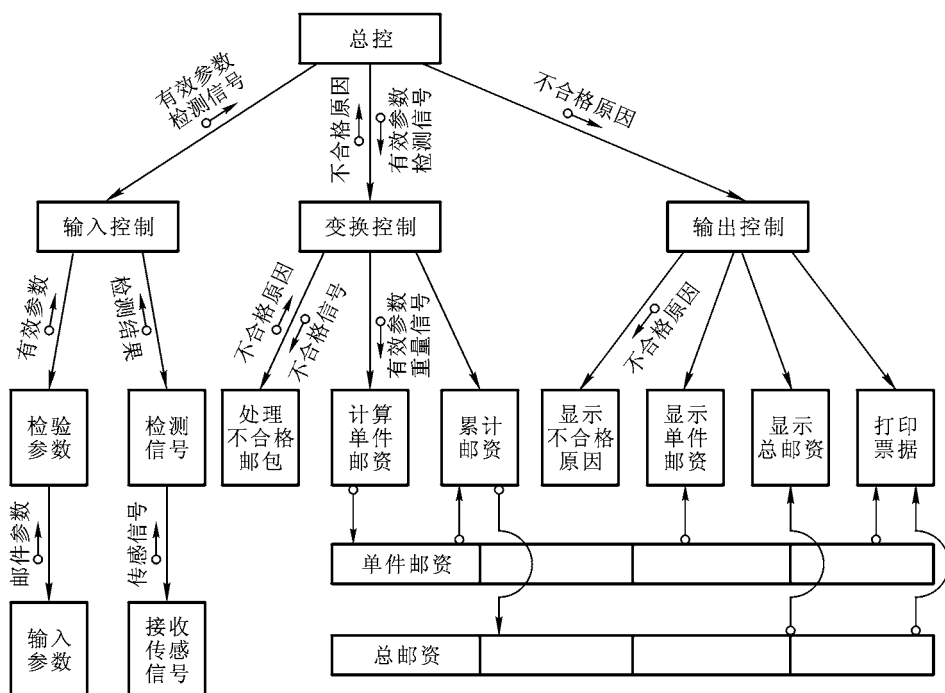


图 5-36 “邮件检测与计价电子仪器系统”软件结构图

## 5.5 数据库结构设计

数据库技术产生于 20 世纪 70 年代初期，从这个时期起，数据库技术经历了层次型、网状型、关系型三种模型。在数据库应用早期，由于层次型、网状型具有的性能优势，它们占据了初期阶段的主流位置。但从 20 世纪 80 年代起，随着计算机性能的迅速提高，关系型数据库所

具有的数学计算方面的优势受到了重视，它逐渐成为主流数据库，并得到了层次型、网状型不曾有过的更加广泛的应用。例如 Oracle、SQL Server 这些耳熟能详的通用大型数据库都是关系型数据库。

数据库就是与特定主题或目标相联系的信息的集合，例如人事数据库、工资数据库等。数据库的作用是能够为软件系统提供后台数据存储与运算。许多应用系统需要依赖数据库提供数据服务，尤其是一些信息管理系统，则更是以数据库为中心进行部署。

与一般数据文件比较，数据库具有以下方面的优越性：

(1) 数据库具有特殊的数据存储结构，例如，关系型数据库中的表结构，更加适宜对数据进行有效的组织、存储和检索。

(2) 数据库具有更加完善的数据完整性约束机制，例如，可以通过设置和字段相关的主键、外键，从而建立起数据表集之间的关联；可以设置对数据字段、记录的检验规则，以限制数据存储范围。

(3) 数据库能够实现数据存储结构与数据表现形式的有效隔离，例如，可以通过数据视图而获得对数据表集更加有效的应用组合。

本节将简要介绍数据库的结构设计，并将从逻辑结构设计和物理结构设计这两个方面分别给予说明。

### 5.5.1 逻辑结构设计

需求分析中已建立了有关数据库的数据关系模型。但是，数据关系模型是基于对用户应用域的分析而构造的，是一个有关现实数据环境的数据库概念模型。显然，这种接近于现实世界的概念模型与计算机世界之间的距离太大了，不能够直接向数据库实现过渡。因此，为了方便数据库的创建，需要将数据库概念模型进行转换，建立一种更加接近计算机世界的数据库模型。

概要设计中需要建立的有关数据库的逻辑结构，就是一种与计算机世界更加接近的数据模型，它提供了有关数据库内部构造的更加接近于实际存储的逻辑描述，因此能够为在某种特定的数据库管理系统上进行数据库物理创建提供便利。

#### 1. 设计数据表

在关系型数据库中，数据是以数据表为单位实现存储的。因此，数据库逻辑结构设计首先需要确定的就是数据库中的诸多数据表。

可以按照以下规则从数据关系模型中映射出数据库中的数据表来。

(1) 数据关系模型中的每一个实体应该映射为数据库逻辑结构中的一个数据表。另外，实体的属性对应于数据表的字段，实体的码对应于数据表的主键。

(2) 数据关系模型中的每一个  $n:m$  关系也应该映射为数据库逻辑结构中的一个数据表。另外，与该关系相连的各实体的码以及关系本身的属性，应该映射为数据表的字段；而与该关系相连的各实体的码，则需要组合起来作为关系数据表的主键。

(3) 数据关系模型中的每一个  $1:n$  关系可以映射为一个独立的数据表（映射规则类似  $n:m$  关系）。但在更多情况下，这个  $1:n$  关系则是与它的  $n$  端对应的实体组合起来映射为一个数据表。当  $1:n$  关系是与  $n$  端对应实体合并组成数据表时，组合数据表的字段中需要含有 1 端实体的码属性。

(4) 数据关系模型中的每一个 1:1 关系可以映射为一个独立的数据表,也可以与跟它相连的任意一端或两端的实体合并组成数据表。实际上,两个依靠 1:1 关系联系的数据表可以设置相同的主键,为了减少数据库中的数据表的个数,往往将它们合并为一个数据表。合并方法是将其中的一个数据表的全部字段加入到另一个数据表中,然后去掉其中意义相同的字段(包括意义相同但名称不同的字段)。

图 5-37 是一个用于描述教师、课程、学生三者之间关系的数据关系模型图。可以按照上述规则对这个数据关系模型进行映射,由此可以产生出以下数据表结构:

教师(教师编号, 姓名, 性别, 职称, 学历)

课程(课号, 课名, 计划课时, 学分)

学生(学号, 姓名, 性别, 专业, 班级)

讲授(教师编号, 课号, 实际课时)

学习(学号, 课号, 成绩)

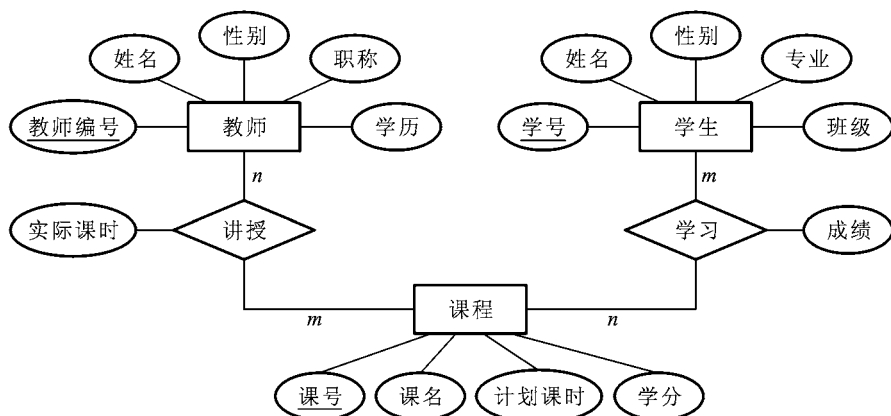


图 5-37 数据关系模型

## 2. 规范数据表

从数据关系模型映射出来的数据表是直接建立在用户应用域基础上的数据表。实际上,同一个数据关系模型可以有多种不同的数据表组合。为了使数据库逻辑结构更加科学合理,设计过程中,一般还需要按照关系数据库规范原理对数据表进行规范化处理,由此可消除或减少数据表中存在的不合理现象,例如数据存储冗余、数据更新异常。

关系数据库规范原理是基于数据冗余程度提出的,包含第一范式(1NF)、第二范式(2NF)、第三范式(3NF)、BC范式(BCNF)、第四范式(4NF)和第五范式(5NF)。其中,第一范式规范化程度最低,数据表内部多余联系最多,数据冗余最大;第五范式规范化程度最高,数据表内部几乎没有多余的联系,数据冗余最小。

显然,通过提高数据表范式级别可以降低数据表中的数据冗余,并可减少由于数据冗余造成的数据更新异常。但是,为了提高数据表范式级别,就需要清除数据表内部的多余联系,这就需要对数据表进行分解。

值得注意的是,对数据表的分解大都会使对数据的查询操作复杂起来(例如多表联接操

作), 由此会降低数据查询性能。

在数据库实际应用中, 为了既能使数据冗余与数据更新异常现象有所减少, 又能使数据查询性能不会出现显著下降, 大多选用第三范式作为设计优化依据。

下面是对数据表中第一范式、第二范式和第三范式的描述。

#### (1) 第一范式

数据表中的每一个字段值都必须是不可再进行分割的原子数据。

#### (2) 第二范式

满足第一范式条件, 而且已经消除数据表中可能存在的非关键字段对关键字段集中个别字段的部分依赖关系。也就是说, 每一个非关键字段都只能由整个关键字段集决定, 而不能由关键字段集中的个别字段决定。

#### (3) 第三范式

满足第二范式条件, 而且已经消除数据表中可能存在的非关键字段之间的传递依赖关系。也就是说, 每一个非关键字段都只能由关键字段集决定, 而不能由非关键字段集决定。

一般情况下, 如果数据表中的数据需要经常更改, 这个数据表可以选用第三范式, 甚至 BC 范式。但如果数据表中的数据不会更改或很少更改, 却经常需要查询, 并且要求有较高的查询性能, 则可以选择第二范式, 甚至第一范式。

### 3. 关联数据表

关联数据表就是将数据关系模型中数据实体之间的关系在数据库逻辑结构中明确体现出来, 它们将作为建立数据表之间参照完整性规则的依据。

图 5-38 是数据库的逻辑模型图, 其中的连线表示了数据表之间的关联, 连线带箭头一端为主表, 另一端为从表。其中, 标记 PK 表示主表中的主键, 标记 FK 表示从表中的外键。主表与从表的关联就建立在主表的主键字段集和从表的外键字段集之间。

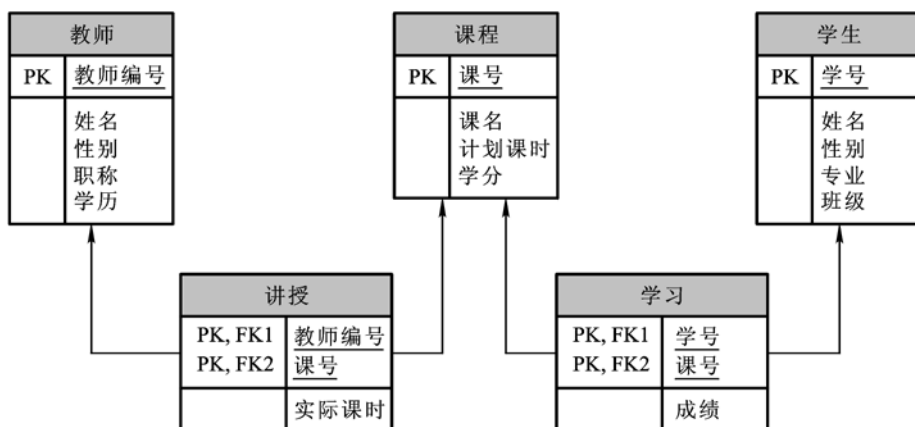


图 5-38 数据库逻辑结构

### 4. 设计数据视图

数据视图也称做虚表, 原因在于数据视图与数据表一样, 都可以将数据以记录集合形式表现出来。但是, 数据视图是面向系统前端应用的不同用户的局部逻辑层。因此, 它与面向系统



后台全局数据存储的数据表也就有着许多不同之处。

一般说来,数据表具有相对稳定的存储结构,它所存储的数据是具体的,并最终会以物理形态在数据库中出现。然而,数据视图却是面向用户的外模式,它并不涉及具体的数据存储,而只是嵌入在数据库中的 SQL 查询操作,因此可以根据用户的应用需要进行比较灵活的数据组合,满足不同用户的数据应用需要。

数据视图的作用是能够使数据表现与数据存储之间进行有效的逻辑隔离。实际上,数据视图为数据库前端应用提供了许多便利。例如,可以使用一些更符合前端用户习惯的数据名称,可以根据用户对数据的特殊需要进行专门的数据视图定义,可以使数据库中的数据具有更高的安全性,可以简化前端程序员对数据库的复杂编程等。

### 5.5.2 物理结构设计

数据库最终是要存储在物理设备上的。数据库在物理设备上的存储结构与存取方法称为数据库的物理结构。为一个给定的逻辑数据模型选取一个最适合于应用的物理结构的过程,就是数据库的物理结构设计。

数据库物理结构设计总是需要依赖于给定的计算机系统,例如,所选用的 DBMS 的特点,需要开发的应用系统对处理频率和响应时间的要求等。

#### 1. 数据存储结构

在确定数据库中数据的存储结构时,需要综合考虑数据存取时间、存储空间利用率和维护代价这三方面的因素。并且,这三个方面还常常是相互矛盾的,例如,消除一切冗余数据虽然能够节约存储空间,但往往会导致检索代价的增加,因此必须进行权衡,选择一个折中方案。

为了提高系统性能,有必要根据数据应用情况将易变部分与稳定部分、经常存取部分和存取频率较低部分分开存放。例如数据库数据备份、日志文件备份等,由于只在故障恢复时才使用,而且数据量很大,可以考虑存放在磁带上。

#### 2. 数据索引与聚集

为了提高对数据表中数据的查询速度,可以在数据表的字段或字段集上建立索引。需要注意的是,索引虽然可以提高查询速度,但索引却需要占用磁盘空间,并且会降低数据更新速度。因此,对于是否设置索引,往往需要根据实际应用进行权衡。如果数据需要频繁更新或磁盘空间有限,则不得不限制索引个数。

许多关系型 DBMS 还提供了聚集索引功能,与一般索引比较,它能够带来更高的查询效率。但必须注意的是,聚集索引只能提高对某些特定字段的查询性能,而且会带来更大的维护开销和存储空间消耗。因此,只有在聚集字段是最主要的查询字段时,才有建立聚集的必要。

#### 3. 数据完整性

为了使数据库中的数据更加便于维护,还需要在数据库中建立数据完整性规则,包括实体完整性和参照完整性。

实体完整性是指数据库对数据表中记录的惟一性约束。为了使数据表具有实体完整性,需要在数据表中设置主键,由此可确保数据表中的每一条记录都是惟一的,也就是说不会出现重复记录。

数据库参照完整性则是指建有关联的数据表之间存在的“主表”对“从表”的一致性约束。

由此可使从表中外键字段的取值能够受到主表中主键字段取值的限制。例如，只有当主表的主键字段中存在该值后，从表的外键字段才能取用该值；并可使得当主表中主键字段值更改时，从表中对应的外键字段值可以自动同步更新，或使得当主表中有记录删除时，从表中外键字段值和主表中主键字段值相同的相关记录也被一同删除。

## 小 结

### 1. 设计过程与任务

概要设计中首先需要进行的是系统构架设计，然后是软件结构、数据结构等方面的设计。主要有以下几个方面的设计任务：制定规范、系统构架设计、软件结构设计、公共数据结构设计、安全性设计、故障处理设计、可维护性设计、编写文档、设计评审。

### 2. 系统构架设计

#### (1) 集中式结构

集中式系统由一台计算机主机和多个终端设备组成。其具有非常好的工作稳定性和安全保密性。但系统建设费用、运行费用比较高，灵活性不够好，结构不便于扩充。

#### (2) 客户机/服务器结构

客户机/服务器结构依靠网络将计算任务分布到许多台不同的计算机上，但通过其中的服务器计算机提供集中式服务。其优越性是结构灵活、便于系统逐步扩充。

#### (3) 多层客户机/服务器结构

- 两层结构：将信息表示与应用逻辑处理都放在了客户机上，服务器只需要管理数据库事务。
- 三层结构：将两层结构的客户机上的容易发生变化的应用逻辑部分提取出来，并放到一个专门的“应用服务器”上。
- B/S 结构：是 Web 技术与客户机/服务器结构的结合。其优点是不需要对客户机进行专门的维护。

#### (4) 组件对象

分布式结构通过组件进行计算分布。它依赖于对象中间件建立，具有灵活的构架，系统伸缩性好，能够给系统的功能调整与扩充带来便利。

### 3. 软件结构设计

软件结构设计是对组成系统的各个子系统的进一步分解与规划。主要设计内容有：确定模块元素、定义模块功能、定义模块接口、确定模块调用与返回、进行结构优化。

#### (1) 模块概念

- 模块化：使用构造程序，可使软件问题简化。
- 抽象化：概要设计中的模块被看成是一个抽象化的功能黑盒子。
- 信息隐蔽：每个模块的内部实现细节对于其他模块来说是隐蔽的。

#### (2) 模块的独立性

软件系统中每个模块都只涉及自己特定的子功能，并且接口简单，与软件中其他模块没有过多的联系。一般采用耦合和内聚这两个定性的技术指标进行度量。

耦合用来反映模块相互关联程度，模块间连接越紧密，耦合性就越高。内聚用来反映模块内元素的结合程度，模块内元素结合越紧密，则内聚性就越高。为提高模块独立性，要求模块高内聚、低耦合。

耦合形式由低至高是：非直接耦合、数据耦合、控制耦合、公共耦合、内容耦合。

内聚形式由低至高是：偶然内聚、逻辑内聚、时间内聚、过程内聚、通信内聚、顺序内聚、功能内聚。

### (3) 设计建模

- 软件结构图：由 Yourdon 于 20 世纪 70 年代提出，被广泛应用于软件结构设计中，能够有效说明软件中模块之间的调用与通信。
- HIPO 图：由美国 IBM 公司推出。其中，H 图用于描述软件的分层调用关系，作用类似软件结构图，IPO 图用于说明描述模块的输入—处理—输出特征。

### (4) 软件结构优化

主要优化设计原则有：使模块功能完整、使模块大小适中、使模块功能可预测、尽量降低模块接口的复杂程度、使模块作用范围限制在其控制范围之内、模块布局合理。

## 4. 面向数据流的结构设计

### (1) 变换分析

软件结构由输入、变换和输出三个部分组成。

### (2) 事务分析

软件结构由接收事务与事务活动两个部分组成。

### (3) 混合流分析与设计

软件系统是变换流与事务流的混合。对于这样的系统，通常采用变换分析为主、事务分析为辅的方式进行软件结构设计。

## 5. 数据库结构设计

### (1) 逻辑结构设计

- 设计数据表
- 规范数据表
- 关联数据表
- 设计数据视图

### (2) 物理结构设计

- 数据存储结构
- 数据索引与聚集
- 数据完整性

## 习 题

1. 概要设计中的主要任务有哪些？
2. 与集中式结构相比，客户机/服务器结构具有哪些方面的优越性？
3. 说明三层客户机/服务器结构中应用服务器的作用。

4. 说明 B/S 结构中 Web 服务器的作用。

5. 说明组件对象分布式结构的特点。

6. 一家跨区域连锁销售企业需要开发一个“物流配送系统”，试说明其适合选择哪种结构的系统构架？

7. 说明概要设计中模块抽象化的作用。

8. 说明模块内部信息隐蔽的作用。

9. 上级模块 A 在调用下级模块 B 时需要返回一个标记 X，用于模块 A 中分支语句的判断条件。这时的模块 A 与模块 B 之间是什么耦合关系？

10. 模块 A 是写文件模块，需要向文件 F 写数据；模块 B 是读文件模块，需要从文件 F 读数据。这时的模块 A 与模块 B 之间是什么耦合关系？

11. 一些通用菜单模块属于哪种内聚形式？

12. 模块 A 是数据查询模块，其模块内部包括输入查询条件、连接数据源、打开查询记录集、显示查询结果。该模块属于哪种内聚形式？

13. 某自动阅卷系统数据流图如图 5-39 所示。该系统能够从卡片读入考生登记和考生答卷，能够对考卷评分产生考生成绩，并能够根据考生登记数据和考生成绩数据产生成绩单数据，由此打印出成绩单。请按面向数据流的结构映射方法，设计该系统的软件结构。

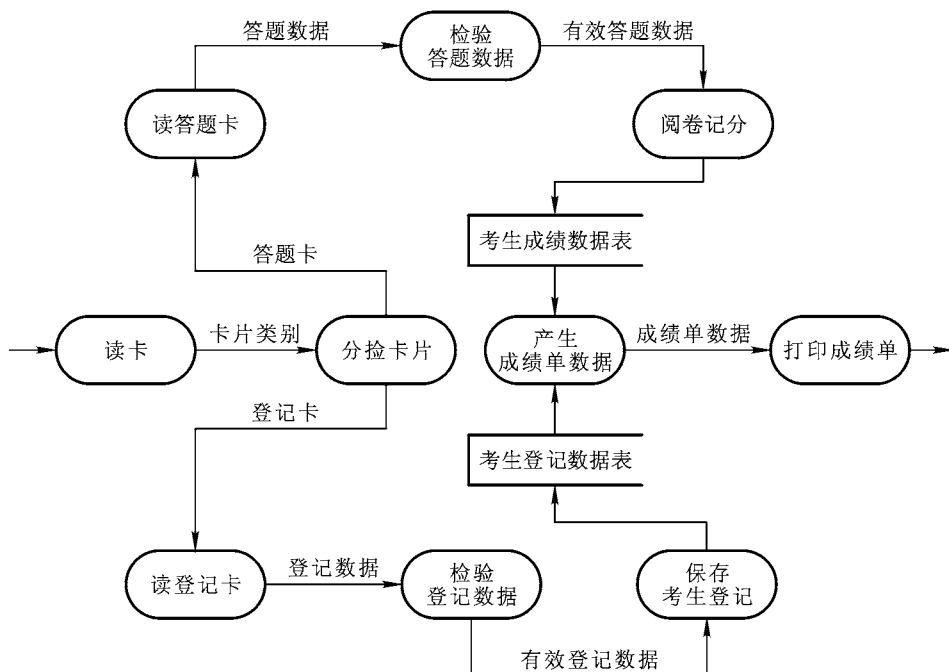


图 5-39 “自动阅卷系统”数据流图

14. 某银行储蓄系统的数据流图已在第 4 章习题 7 中要求画出。请根据你所画出的数据流图设计该系统的软件结构。

15. 某图书馆图书借阅系统的数据关系模型如图 5-40 所示。请根据该数据库概念模型设计该数据库逻辑结构。

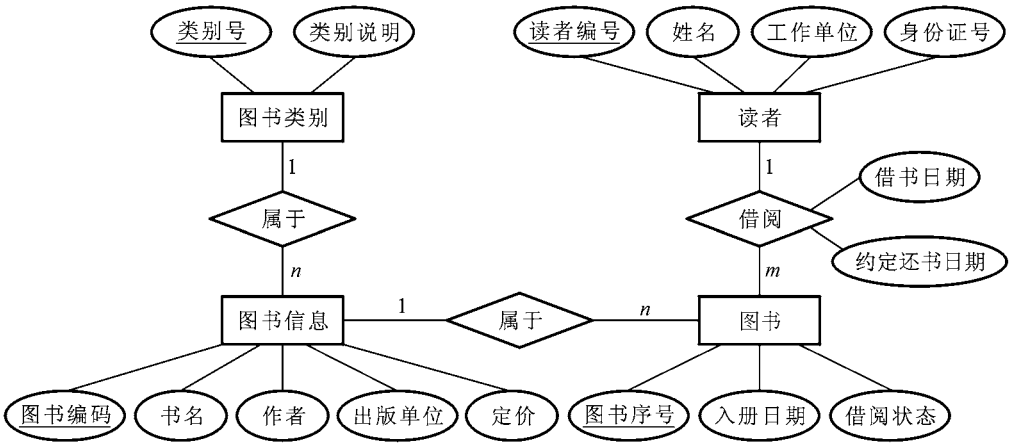


图 5-40 图书借阅系统数据关系模型

## 第 6 章 面向对象分析与设计

前面章节对于软件分析与设计的讨论主要基于结构化方法，本章将专门讨论面向对象分析与设计方法，并采用 UML 建模语言对面向对象问题进行描述。

### 6.1 面向对象方法学

计算机处理问题的方法总是依赖于程序设计语言。例如汇编语言，它是面向机器的；因此，其处理问题的方法就与机器的关系非常接近，采用了对内存地址直接指向的方式来操作有待加工的数据。

计算机编程技术在不断进步，其特征之一就是不断地由接近机器转变为接近于人。因此，20 世纪 60 年代出现了高级编程语言，采用了面向人所面临的问题加工过程的处理流程。20 世纪 70 年代，结构化的模块概念被引入到了程序设计语言中，其更进一步地改善了人与计算机之间的关系，使得软件可以基于功能模块来构造，例如过程、函数。但是，结构化功能模块中的待加工数据仍需要通过模块接口或环境外部提供。

自 20 世纪 80 年代以来，诸多面向对象编程语言诞生了，例如 C++、Java 等。面向对象编程语言对问题的处理是基于对象进行的，而对象则是一个既具有数据元素，又具有行为能力的高度集成的模块单元，比较起结构化方法中的过程、函数这些功能模块来，对象模块更加接近于现实世界中的实体。

面向对象技术涉及面向对象分析（OOA）、面向对象设计（OOD）和面向对象编程实现（OOP）这三个方面的问题。面向对象技术的出发点是尽可能地模拟现实世界，由此使开发软件的方法与过程尽可能地跟人对世界的认识保持一致。

可以说，面向对象技术使人本精神在计算机这个机器世界中获得了更加有效的发挥，并在面向用户的应用系统开发中表现出了显著的优势，成为了这个领域的主流开发工具。正是面向对象软件技术的广泛应用，由此推动了面向对象的软件工程方法学的发展。

#### 6.1.1 面向对象方法的基本概念

##### 1. 类

在面向对象方法中，类是一个静态的模块单位，其作用是为创建对象实例提供一个模板。因此，可以把“类”看作生产对象零件的模具。

类还是一个群体概念，具有同一类事物的共同特征。例如可以把各种不同型号不同颜色的具体的汽车的特性归纳起来，产生出来有关“汽车”的类概念，它具有所有汽车的共同特征。

面向对象方法中的对象是数据与对数据的操作的封装，因此用于产生对象的类，也就需要从数据特征与行为特征这两个方面对其进行描述。其中，类的数据特征用“属性”表示，而类的行为特征用“操作”表示。

图 6-1 是用于描述“汽车类”的类图，其类名是“汽车”，这是它的标识符，属性有“类别”、“型号”、“生产厂家”、“出厂日期”，操作有“驱动”、“制动”。

2．类中属性、操作与方法

类的属性如同类自身一样，具有抽象、无值的特征，只有在通过类产生出具体对象之后，属性才能够具有具体的数值。这也即是说，只有在类模块被程序引用并产生出对象实例之后，类中属性才会在对象实例中体现出具体数据。

操作用于定义类的行为特征，若所定义的操作具有公共特性，则这个操作就能成为通过类而产生的对象提供给外部的功能操作接口，能够被对象以外的其他对象调用。

方法用于说明操作的实现细节。为了使由类产生的对象能够有效执行类所定义的操作，必须在类中建立方法。实际上，类中方法也就是类所具有的内部处理过程，它对应于操作接口，用于提供服务细节。

3．类的继承性

类的继承性的获得源于将类组织成一个具有等级层次结构的系统。一个类的上层可以有父类，下层可以有子类，由此以来，上级父类能够把自己的属性、操作传递给下级子类。例如图 6-2 中的“学生”和“教师”，它们除了具有自己特有的属性、操作之外，还继承了其父类“学校成员”所具有的属性与操作。

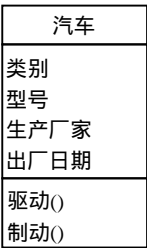


图 6-1 “汽车类”的类图

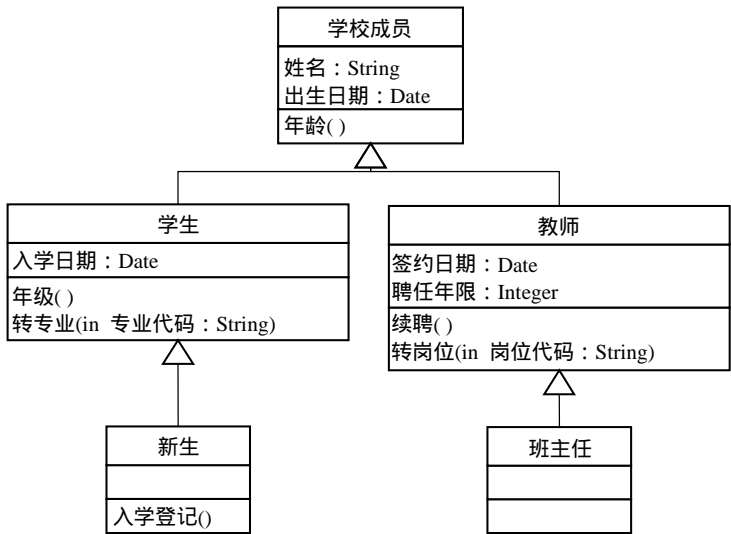


图 6-2 类的继承

类的继承具有传递性，这使得类能够继承它所在的类等级中和在它上层的全部基类的所有描述。例如图 6-2 中的“新生类”，由于“学生类”继承了“学校成员类”，而“新生类”继承了“学生类”，因此，“新生类”也就继承了“学校成员类”。

在面向对象软件技术中，继承使子类能够自动地共享父类中定义的数据与操作，使得诸多

相似的对象可以共享程序代码和数据结构，由此可以大大减少程序中的冗余信息。

面向对象软件技术的许多强有力的功能和突出的优点，也都来源于类的继承性。可以使用从原有父类派生出新的子类的办法来修改软件，由此可以使对软件的修改变得更加方便。例如改变类原有操作的算法，这时并不需要改变类原有的代码，而只是新建一个派生子类，然后在该派生子类中建立一个与父类方法同名而算法不同的方法。

实际上，通过类的继承性还可使软件获得更加有效的重用。当需要开发一个新的应用系统时，可以通过继承而使原有的相似系统的功能获得利用；或者从类库中选取需要的类，然后再派生出新的子类，以实现新系统的新功能。

#### 4. 类的多态性

多态性是指子类对象可以像父类对象那样使用，它们可以共享一个操作名，然而却有不同的实现方法。也就是说，不同层次中的类，可以按照各自的需要分别采用不同的方法实现这个共同的操作。

在 C++语言中，多态性是通过虚函数来实现的。在类等级不同层次中，可以说明名称、参数特征和返回值类型都相同的虚拟成员函数，但不同层次的类中的虚函数的实现算法却各不相同。虚函数机制使得程序员能在一个类等级中使用相同函数的多个不同版本，在运行时才根据接收消息对象所属于的类，决定到底执行哪个特定的版本。

多态性机制不仅增加了面向对象软件系统的灵活性，而且显著提高了软件的可重用性和可扩充性。当需要扩充系统功能或在系统中增加新的实体类时，只要派生出与新的实体类相应的新的子类，并在新派生出的子类中定义符合该类需要的虚函数，而无须修改原有的程序代码。

#### 5. 对象

对象是属性与操作的封装体，是类模块实例化的结果，图 6-3 是由图 6-2 中的“班主任类”、“学生类”产生的两个对象实例。

班主任实例 1：班主任	学生实例 1：学生
姓名：String = 刘华	姓名：String = 王燕
出生日期：Date = 1970.6.16	出生日期：Date = 1981.6.18
签约日期：Date = 1998.7.20	入学日期：Date = 2001.9.8
聘任年限：Integer = 5	

图 6-3 由类产生的对象实例

对象依靠对象名称进行标识，依靠属性值反映其状态，依靠操作向外界提供服务。对象的属性值只能由该对象的操作来改变，每当需要改变对象的状态时，可以由其他对象向该对象发送消息，然后对象响应消息，并按照消息模式执行与其匹配的方法。

对象是动态的，并具有以下一些基本特点：

(1) 对象以数据为中心。对象的操作是围绕其数据所需进行的处理而设置的，操作结果往往与对象当时的属性值有关。

(2) 对象以操作为功能接口。不能从外部直接加工对象的私有数据，而必须通过对象的公共接口向它发送消息，请求它执行某个操作，处理它的私有数据。

(3) 对象具有数据封装的特征。可以将对象看作是一个黑盒子，它的私有数据被封装在盒



子内部，对外是隐蔽的。为了使用对象内部的私有数据，只需知道数据的取值范围和可以对数据施加的操作，而无须知道该数据的具体结构以及实现操作的具体方法。

(4) 对象能够并行工作。实际上，不同的对象能够各自独立地处理自身的数据，彼此之间只是通过发送消息实现通信。

(5) 对象具有较强独立性。对象内部各种元素彼此结合紧密，具有较高的内聚性。由于对象的功能被封装在对象内部，因此对象与外界的联系比较少，其耦合性低。

## 6. 消息

消息是指对象之间的通信，而向某个对象发送消息，则要求这个对象执行它的某个操作。例如，“班主任实例1”对象需要向“学生实例1”对象发送转专业到“软件工程专业”的消息，假如“软件工程专业”的专业号是：J006，则该消息可以通过图6-4表达。

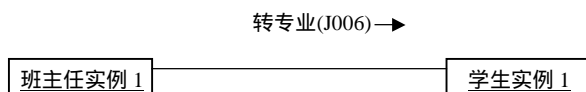


图6-4 班主任对象向学生对象发送消息

通常情况下，一个消息由以下三个部分组成：

- (1) 接收消息的对象名。
- (2) 需要执行的操作名，也叫消息名。
- (3) 向需要执行的操作的形式参数传递实际参数值。

### 6.1.2 面向对象方法具有的优越性

#### 1. 与人所面对的现实世界更加接近

面向对象软件技术以对象为核心，提供了对现实世界更加完整地描述，涉及数据、行为和通信等。在面向对象方法中，计算机观点已被淡化了，现实世界模型成为构造系统的最重要依据，并鼓励开发者，在软件开发过程中能够更多地使用用户的应用领域中的概念去思考问题。

实际上，面向对象的软件开发过程从始至终都在考虑如何建立问题领域对象模型，从对问题领域进行自然的对象分解，到确定需要使用的类、对象，再到在对象之间建立消息通道以实现对象之间的联系，等等。

应该说，面向对象方法为开发者提供了更多的与用户交流的机会，并提供了随着对应用系统认识的逐渐深入，而逐步地进行系统设计与实现的可能性。例如，可以先设计出由抽象类构成的系统框架，然后随着认识的深入，再逐步派生出更加具体的派生类。这样的开发过程与人在解决复杂问题时逐步深化的渐进过程更加显得一致。

#### 2. 可使软件系统结构更加稳定

传统方法的开发过程是基于功能分解的，用传统方法所建立起来的软件系统的结构直接依赖于系统所要完成的功能。因此在传统方法之下，若系统功能需求发生了变化，则有可能引起软件结构的整体变动。

与结构化系统构造方法不同，面向对象方法是以对象为中心来构造软件系统。它的基本作法是用对象模拟问题领域中的实体并以对象之间的联系来表现领域实体之间的联系。因此，当

对系统的功能需求改变时，一般仅需要对一些局部对象进行修改，而并不需要改变整个软件结构。例如，可以从已有的类中派生出一些新的子类来，以实现系统功能扩充或修改。

### 3. 软件具有更好的可重用性

软件重用是提高软件生产率的最主要的途径之一。

传统的软件重用技术利用的是标准函数库，但标准函数缺乏必要的“柔性”，大多数的库函数往往只提供最基本、最常用的功能，因此不能适应不同应用场合的不同需要。

比较起传统方法来，面向对象方法在利用可重用的软件成分构造新的软件系统时，具有更大的灵活性。其中，最主要的重用机制是利用了类的继承性，可以通过上级父类派生出下级子类。由父类派生出的子类，不仅可以重用其父类的数据结构和程序代码，而且可以在其父类代码的基础上，方便地进行子类的修改与扩充。

### 4. 软件更加便于维护与扩充

面向对象方法在软件维护与扩充方面所具有的优越性主要体现在以下几个方面：

(1) 软件系统比较稳定。当软件功能或性能需求发生变化时，通常不会引起软件体系发生整体变动，而只需要改变局部类模块，比较容易实现。

(2) 软件比较容易修改。类是独立性更强的模块，一个类的修改一般很少牵扯到另一个类。如果只修改类的内部实现部分，而不修改该类的对外接口，则可以完全不影响软件的其他部分。另外，可以利用类所提供的继承机制进行软件扩充，由于是从已有类中派生出一些新的子类来进行扩充，因此不需要修改软件原有成分。

(3) 软件比较容易理解。在维护软件时，需要对软件的修改部分有深入理解。但传统软件的修改内容往往分散在软件的各个地方，需要了解的面很广，而且软件结构与问题空间结构很不一致。而面向对象软件系统的修改则一般集中在一个或少数几个类模块上面，并且软件系统结构与问题空间结构具有更好的一致性。

## 6.1.3 UML 建模方法

随着面向对象编程技术的广泛应用，面向对象分析设计建模方法学也相应快速发展起来。在 20 世纪 80 年代末期至 90 年代初期，产生出数十种面向对象建模方法。其中，比较有代表性的对象建模方法有：

1986 年 Booch 提出的面向对象分析与设计方法论 (OOA/OOD)。

1991 年 Rumbaugh 提出的对象模型技术 (OMT)。

1994 年 Jacobson 提出的面向对象软件工程方法学 (OOSE)。

应该说，诸多对象建模方法的产生，一方面强有力地推动着面向对象工程方法学的快速增长，但另一方面却也带来了一些思想上的混乱，面对种类繁多的建模方法，软件工作者眼花缭乱、无从选择。

显然，面向对象建模方法经过一段时期的探索，已经发展到了需要进行整合的时候。1995 年，Booch、Rumbaugh 和 Jacobson，这三位对象建模方法学大师，将他们各自的对象建模方法结合到一起，产生了统一的面向对象建模方法——UML 建模。

UML 建模方法的产生对于面向对象方法学而言，具有划时代的意义。1997 年，国际对象管理组织 (OMG) 接受它作为标准建模语言，随后 UML 建模方法得到了更进一步的完善，并在整个

信息技术领域获得了非常广泛的应用。

### 1. UML 模型图的组成

UML 建模语言能够从各个不同的角度对软件系统进行描述，所用到的图形模型如下：

(1) 用例图。涉及参与者、用例等图形元素，用于描述用户与系统之间的交互关系。

(2) 类图。涉及类、接口等元素以及这些元素之间存在关联、泛化、依赖等关系，用于描述系统的静态结构。

(3) 活动图。使用活动图形元素说明系统的高层活动状态与转换，用于说明用例图中每个用例的内部工作流程。

(4) 状态图。涉及状态、事件等图形元素，用于对类元素所具有的各种状态以及状态之间的转换关系进行细节描述。

(5) 序列图。涉及对象、消息等图形元素，其中对象沿横向排列，消息沿竖向排列，用于反映对象通信时传递消息的时间顺序。

(6) 协作图。涉及对象、消息等图形元素，用于对系统在某个工作片段所包含的对象以及对象之间基于消息的通信进行设计说明。

(7) 构件图。使用构件及其构件之间的依赖关系说明系统的物理构造。

(8) 部署图。由客户机、服务器等物理节点组成，用于描述系统的分布式架构。

### 2. UML 建模过程

基于 UML 的建模步骤如图 6-5 所示，包括分析与设计这两个建模阶段。其中分析阶段需要创建的模型有：用例图、活动图、类分析图 and 序列图，设计阶段需要创建的模型有：类设计图、协作图、状态图、构件图和部署图。

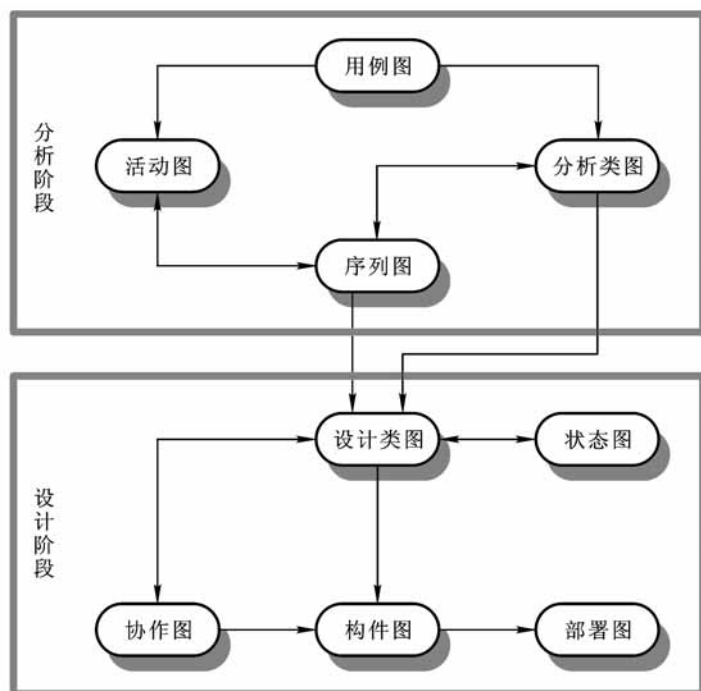


图 6-5 UML 基本建模过程

由于面向对象分析与设计采用了一体化的 UML 建模工具，这使得分析阶段产生的一系列结果不仅成为设计阶段的导入条件，并且诸多结果可以通过设计进行补充并逐步完善。例如类图模型。分析阶段只是建立了类的概念模型，这是一个有关类的数据库模型，主要涉及的是系统中与数据结构有关的实体类，需要确定的是这些实体类的名称、属性等。至于与类有关的操作问题，则被留到了设计阶段加以解决，例如，实体类以外的界面类、控制类的定义，类中数据属性以外的操作、方法的定义等。

基于 UML 的建模过程是一个以增量方式迭代的过程，需要进行多次的反复，图 6-5 中分析类图与协作图之间，设计类图与协作图、状态图之间的双向箭头即表明了这种迭代关系。应该说，迭代作为一种思想已经溶于面向对象分析与设计之中，正是依靠迭代过程与 UML 一体化建模的结合，由此使得分析与设计之间的能够获得有效的无痕过渡与进化，使得软件系统可以经过许多次的分析与设计的交替而不断趋于完善。

## 6.2 面向对象分析建模

面向对象分析建模需要建立的是软件系统的用户领域模型，需要着重了解的是该软件系统的需求概念与术语，其分析内容是现实世界中的实体对象和各对象之间的关系，并不涉及编程概念。

如同第 4 章中介绍的那样，软件分析需要经历由用户需求到系统需求的过渡，并由此为系统设计创造条件。因此，面向对象分析也就需要从面向用户的用例建模开始，并通过活动图、类图和序列图等，分别从系统业务流程、组织结构和行为过程等几个方面对系统进行分析，由此产生出有关系统的分析模型（建模流程见 6.1.3 节中的图 6-5）。

### 6.2.1 用例图

用例图由 Jacobson 提出，并随着他所倡导的 OOSE 而被引入面向对象方法学中。在 UML 建模语言中，用例图被用来描述用户与系统之间的交互关系，说明系统所具有的业务能力和业务流程，能够方便开发者理解用户领域的专有术语和业务内容。

图 6-6 是一个商店售货系统的用例图。其中的“椭圆”符号表示的系统中的用例，“人形”符号表示的系统的参与者，“连线”表示参与者与用例之间的通信。为了使系统从环境中分离出来，其使用了“矩形框”反映系统边界。

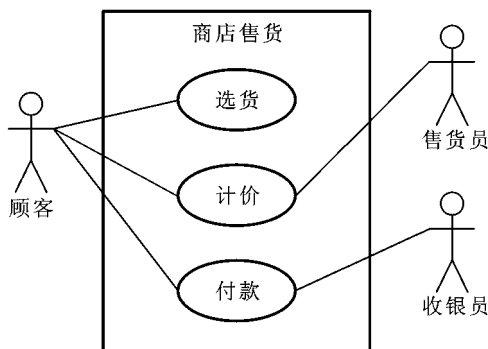


图 6-6 用例图模型

应该说,用例图是一个非常有效地表现系统外部特征的用户需求描述工具,能够直观地描述系统的业务功能和应用接口,并且可以通过用例图对用户需求的捕获以及驱动软件需求过程的进行,促使软件需求不断趋于完善。

## 1. 图形元素特点

### (1) 用例

用例是用例图中最基本的图形元素,代表了组成系统的业务类元,被用来表示软件系统中一个面向用户的业务级功能单元,能够从用户角度出发,在系统的业务层面对系统进行功能分割。

用例图中的用例一般具有以下特点:

- 用例代表某些用户可见的功能,用于表达一个具体的用户目标。
- 用例需要由参与者激活,并能提供确切的值给参与者。
- 用例可大可小,但它必须是对一个具体用户目标的完整体现。
- 用例在以后开发过程中,可以进行独立的功能检测。

### (2) 参与者

用例图中的参与者是指系统之外与系统有关的类,涉及与系统进行交互的人、机器或其他系统等,用于反映系统跟周围环境的关系。

需要注意的是,尽管用例图中的参与者是使用“人形”符号表示,但这并不意味着参与者就一定是一个人,它也可以是系统之外的机器、机构或其他系统等。并且参与者还是一个群体概念,它所体现的是系统之外的环境中的类,代表的是一类能够使用某个功能的人或物,而不是某个个体。例如图6-6中的“顾客”,他可以是张三,也可以是李四,但是张三或李四这样的个体对象不能称为参与者。事实上,一个具体的人、机构、部门等,在系统中可以担任多种不同的参与者角色。

### (3) 通信

用例图使用线段把参与者与用例连接起来,以反映两者之间的通信。其中,单个参与者可与多个用例联系;反过来,一个用例也可与多个参与者联系。但对于同一个用例,尽管有许多参与者与它联系,然而不同的参与者会有不同的作用。例如图6-6中的“顾客”与“收银员”,尽管都与付款联系,但作用却根本不同。

## 2. 用例之间的关系

用例是面向用户的业务类元,具有类的一般特征。因此,用例之间的关系可以使用类所具有泛化关系符进行描述。用例之间主要有“扩展”和“使用”这两种特殊的泛化关系,下面分别给予说明。

### (1) 扩展关系

当某个基本用例由于需要附加一个用例来扩展或延伸其原有功能时,附加的扩展用例与原有的基本用例之间的关系就体现为扩展关系。

在UML用例图中,可以使用带有《extends》说明的泛化关系表示“扩展关系”。如图6-7所示,其中的“下发会议通知”与“下发紧急会议通知”之间,“处理付款”与“网上支付书款”之间,就是扩展关系。

一般说来,扩展用例可以继承原有基本用例的一些功能,同时它又可以具有一些新的特有

的功能，例如图 6-7 中的扩展用例“下发紧急会议通知”对于基本用例“下发会议通知”所具有的继承性。

许多情况下，还可以把系统中那些特殊功能作为扩展用例附在用来表示必须功能的基本用例上，以表示特殊功能与基本功能之间的差别。例如图 6-7 中的扩展用例“网上支付书款”与基本用例“处理付款”，它们之间即具有这样的特点。

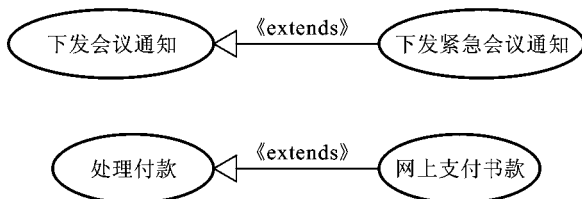


图 6-7 用例之间的“扩展关系”

### (2) 使用关系

当在一个基本用例中使用到了另一个用例时，则这两个用例之间就构成了使用关系。其中，基本用例是一个更大作用范围的用例，它将被使用的用例包含于其中。也可以认为，使用用例的一个实例被引进了基本用例之中。

在 UML 用例图中，可以使用带有《users》说明的泛化关系表示“使用关系”。如图 6-8 所示，其中的基本用例“产生定单”、“查询定单”与被使用的用例“验证会员身份”之间就是使用关系。

一般说来，如果在多个用例中有一些共同的功能，则可以把这些共同的功能提取出来单独构成为一个用例，而其他用例则可以通过“使用关系”共同使用这个用例。

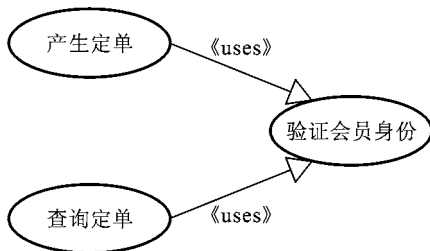


图 6-8 用例之间的“使用关系”

### 3. 用例图举例

用例图所要建立的是用户业务模型，其采用了图形元素对用户业务需求进行可视化表述，因此有利于由用户需求到系统需求的过渡。

用例建模是直接面向用户的，其建模可以以用户针对软件系统的需求陈述为基本依据。其中用户需求陈述中有关系统的业务边界、使用对象等，是构造系统用例模型的基本元素。一般说来，可以按照以下步骤建立系统用例模型：

(1) 从几方面识别系统参与者，包括：需要从系统中得到服务的人、设备和其他软件系统，

以及系统中诸多功能的操作者、系统维护者、系统管理者等。

(2) 分析系统业务边界或系统参与者对于系统的基本业务需求, 可以将其作为系统的基本用例。

(3) 分析基本用例, 将基本用例中具有有一定独立性的功能, 尤其是具有公共行为特征的功能分解出来, 将其作为新的用例供基本用例使用。

(4) 分析基本用例功能以外的其他功能, 将其作为新的用例供基本用例进行功能扩展。

(5) 分析并建立参与者与用例之间存在的通信关系。

为说明上述用例建模步骤, 本节将介绍“网上图书销售系统”用例图的建模过程, 下面是该系统的用户需求陈述:

(1) 某图书销售机构准备开发“网上图书销售系统”, 以方便顾客通过互联网进行购书活动。

(2) “网上图书销售系统”考虑采用会员制进行顾客管理与图书销售。顾客需要先注册, 然后才能定购图书。顾客在进行会员注册时需要设置登录密码, 并输入用于购书联系个人信息, 包括姓名、详细收货地址、邮政编码、Email 地址、电话等。在顾客成为会员后, 顾客会获得一个由系统提供的标识其消费身份的会员标识码。

(4) 在顾客登录进入“网上图书销售系统”以后, 顾客可以通过一个 Web 页面查看图书品种信息, 顾客可以在此选购图书, 由此可将需要购买的图书添加到自己的购书定单细目中, 并可通过购书定单细目窗设置每种图书的定购量, 系统则能根据图书定购量、定价和折扣率等, 进行自动计价。

(5) 顾客购书时需要向图书销售机构提交购书定单, 它是顾客购书时的合同凭据, 其内容包括定单编号、总金额、定单细目(清单)、收货人信息、付款方式、交货期限等。在产生定单过程中, 系统需要对顾客的会员身份进行验证, 并需要以顾客的会员信息作为默认收货人信息交顾客确认或修改。

(6) 顾客可以查询自己的定单状况, 并可以在产生发货通知以前取消定单。在查询定单时, 系统需要对顾客的会员身份进行验证。

(7) 在定单成功提交以后, 图书销售机构需要对定单进行处理, 如向图书配送机构发出发货通知。图书销售机构也有权取消定单。

(8) 系统需要对顾客付款情况及其账务进行有效管理, 并能够支持网上支付。

从上述需求陈述中可以发现以下元素:

(1) 参与者

- 顾客
- 图书销售机构
- 图书配送机构

(2) 基本用例

- 注册会员
- 选购图书
- 产生定单
- 查询定单

- 处理定单
- 处理付款

在基本用例确定以后，接着可以根据需求陈述分析与基本用例有关的功能，由此可以发现用例之间的扩展关系、使用关系。

#### (1) 扩展关系

- 选购图书 设置定单细目
- 选购图书 检索图书信息
- 处理定单 通知发货
- 处理付款 网上支付书款

#### (2) 使用关系

- 产生定单 验证会员身份
- 查询定单 验证会员身份
- 查询定单 取消定单
- 处理定单 取消定单

将上述参与者、用例加入到用例图中，并建立参与者与用例之间的通信，由此可以获得如图 6-9 所示的“网上图书销售系统”的用例图模型。

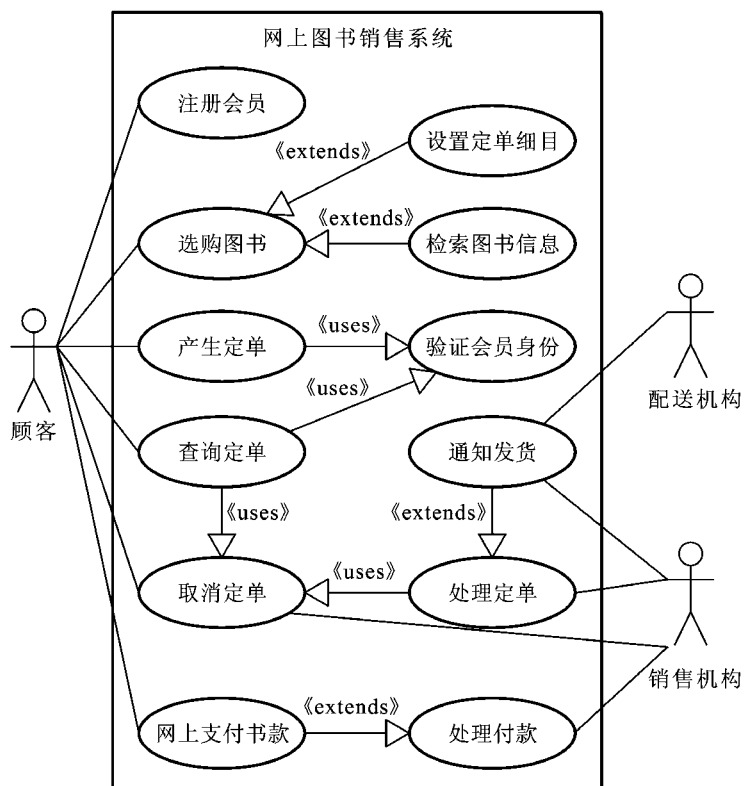


图 6-9 “网上图书销售系统”用例图模型



6.2.2 活动图

活动图是一种行为模型，主要用于描述用例图中用例的内部活动状态与活动转换过程，以获得对用例的交互行为与工作流程的细节说明。涉及用例有哪些内部活动，活动结果是什么（活动状态转换），以及在何时（活动序列）何处发生（泳道）等。

活动图所使用的图形标记与 4.4.4 节中的状态图的图形标记基本相同。不同之处是状态图中的状态具有静态特征，需要通过外部事件才能发生状态转换。而活动图中的状态则具有动态特征，有内部动作，其状态的改变一般由内部事件驱动。另外，活动图还有一个用纵向矩形表示的“泳道”标记，用于聚合一组活动，可以使活动按服务对象分区。

图 6-10 是“网上图书销售系统”的全局主流事件活动图。

通常情况下，用例图中的每个用例都应该使用活动图描述其内部活动细节，其将作为以后设计时的有效依据。图 6-11 是图 6-9 中的“产生定单”用例的内部活动图。

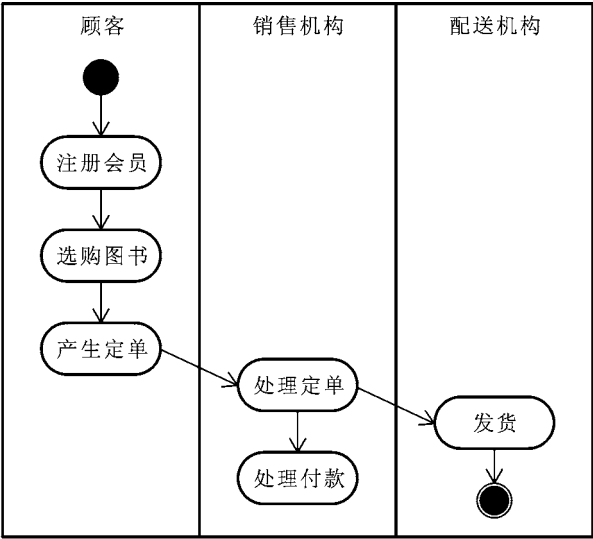


图 6-10 “网上图书销售系统”的全局主流事件活动图

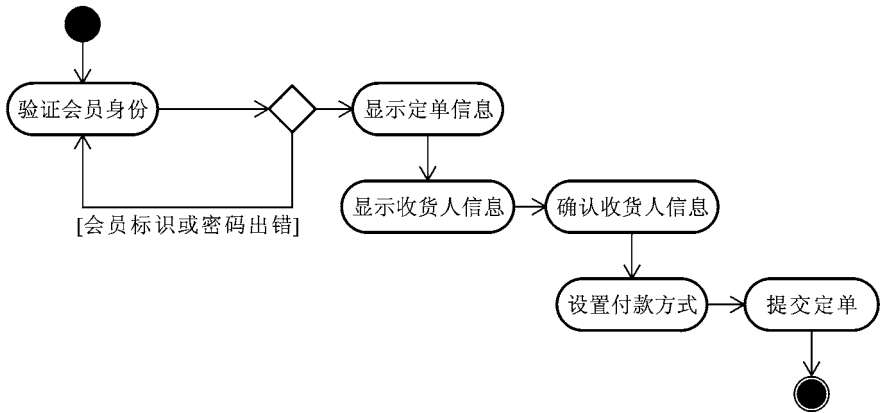


图 6-11 “产生定单”用例的活动图

### 6.2.3 分析类图

UML 中类元素的图形符号在 6.1.1 节中就已经使用了，这是一个分成上、中、下三个区域的长方形符号，分别用于表示类的类名、属性和操作，如图 6-12 所示。类图建模就是基于这个图形元素进行的，其用于描述系统中的类、接口以及它们之间关系，由此建立起系统的静态模型。

应该说，类图是面向对象建模的核心内容，能够作为构造状态图、序列图、协作图和构件图的基础。实际上，类图贯穿于软件分析到软件设计，并能够为软件系统的实现提供依据，面向对象工程方法从分析到设计的无痕迭代，就是依靠类图进行的。

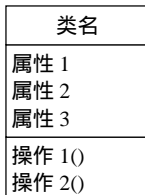


图 6-12 类元素图形符号

#### 1. 定义实体类

需求分析阶段需要获得的是有关类图的概念模型，其主要成分是用于体现现实世界中数据构造的实体类，它们是系统需要的数据成分。需求分析阶段建立类图模型的第一项任务就是发现这些实体类。其中，比较常用的发现类的方法是名词短语法。

名词短语法对于类的发现建立在对用户需求陈述的构词分析上，系统分析人员可以从对用户需求陈述的阅读中寻找名词或名词短语，并把每一个名词或名词短语作为候选类记录下来。

以前面介绍的“网上图书销售系统”的用户需求陈述为例。从该需求文档中可以发现一些与类有关的名词或名词短语，如销售机构、图书品种、顾客、会员、发货地址、定单、定单细目、付款、发货单和配送机构等。

上述名词或名词短语可以看作为构造系统的候选类。接下来需要对它们做进一步的分析，以确定其是不是确实对应于一个实体类。例如其中的图书品种、会员、定单、付款、发货单等，是在图书销售系统之内，并能够作为一个独立的数据实体存在，因此可以比较明确地将它们确认为实体类。而“发货地址”在该软件问题中，它们不是作为一个独立的数据实体存在，而是存在于会员这个实体类之内，因此它们不能作为实体类来对待，而只能是看作为会员类的属性。另外则是“配送机构”，如果系统中只有一个专门的图书配送机构，则该配送机构可以被排除在需要进行操作的实体类之外。但是，若系统中有多个不同的图书配送机构负责送书服务，则图书配送机构也就需要进行选择操作了，因此有必要被作为一个类来对待。

依据上述原则可以分析出“网上图书销售系统”中包含以下实体类：会员、定单、定单细目、图书品种、付款、发货单和配送点，如图 6-13 所示。

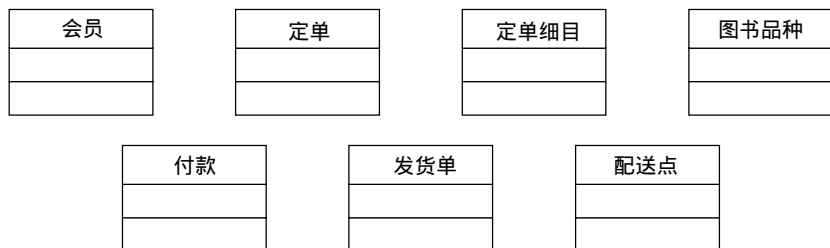


图 6-13 “网上图书销售系统”中的实体类

需求分析时对实体类的定义还包括分析类所具有的属性，这往往依赖于对系统应用领域的熟悉程度。系统分析人员对于应用领域越熟悉，越容易将类应该具有的属性寻找出来。寻找类的属性与寻找实体类可以一同进行，并一同被发现。那些没有必要独立存在数据单位可以考虑作为它所依附的类的属性存在。例如，“网上图书销售系统”中的会员的姓名、发货地址，图书品种的书名、单价、折扣率，定单的定货日期、金额等。

发现类的属性是一个需要反复迭代的过程，需求分析时可以将侧重点放在类的明显的、基本的属性上，其他属性则可以在后面的设计中，甚至系统的功能扩充中逐步发现。

## 2. 确定关系

如前所述，类图由类和它们之间的关系组成。定义了类之后，接着还需要确定类之间的各种关系。类与类之间的关系主要有：关联、泛化和聚集，下面分别给以介绍。

### (1) 关联关系

在用户需求陈述中，类之间的关联往往表现为两个类之间存在某种语义上的联系。例如下面的陈述：若顾客通过互联网购买商品，需要先提供购物定单。这条陈述即反映了“顾客”与“定单”之间所存在的语义上的联系。可以通过在类图中建立关联关系将类之间的联系反映出来（如图 6-14 所示）。

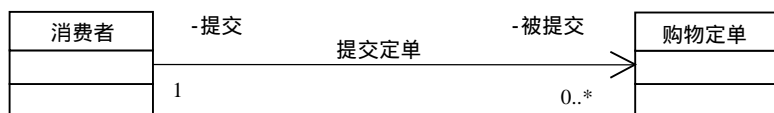


图 6-14 类之间的关联

关联关系一般使用连接两个类的关联线表示。关联线可以提供以下信息：

关联名称：用于表示标记关联，如图 6-14 中的“提交定单”。

关联端名：用于反映类在这一关联关系中扮演了什么角色，如图 6-14 中的“提交”、“被提交”。

关联导向性：是指关联关系只在指定方向上成立，可以使用是一个指向符号表示，例如图 6-14 中的“提交定单”，其导向是由消费者指向购物定单。

关联多重性：表示是由类产生的对象之间存在的数量关系，主要有以下几种数量关系。

1..1 或 1           表示 1 个对象

0..1           表示 0 到 1 个对象

0..\* 或 \*       表示 0 到多个对象

1..\*           表示 1 到多个对象

### (2) 关联限定符

关联限定符通常用于一对多或多对多关联关系中，用于指明如何识别关联关系中另一端的类中的对象，可使多重性由一对多或多对多缩减为一对一或多对一的，如图 6-15 中的“定单编号”。

### (3) 关联类

关联有可能具有自己的属性或操作，对此需要引入一个关联类来进行记录。这时，关联关

系中的每个连接与关联类的一个对象相联系，关联类通过一条虚线与关联连接。例如图 6-16 中的类图模型，其中的“讲授”、“学习”就是关联类。

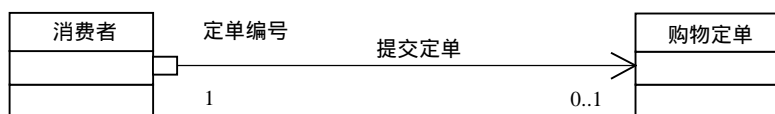


图 6-15 关联限定符

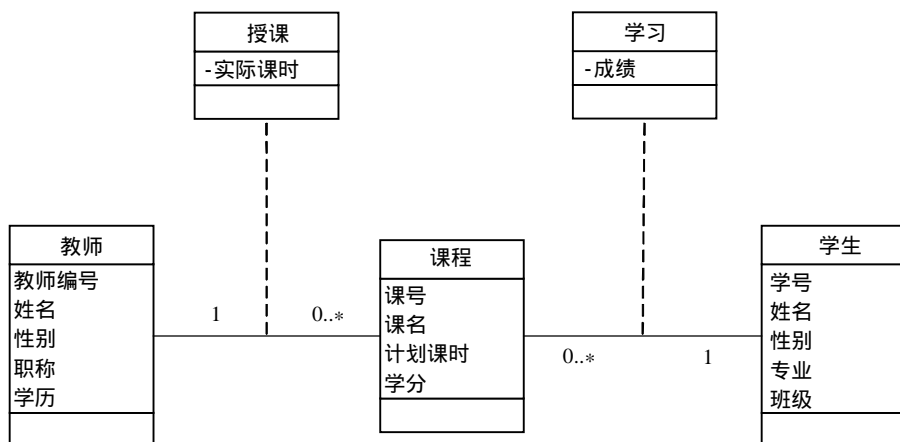


图 6-16 关联类

#### （4）聚集关系

聚集是一种特殊的关联，用于反映类图中具有整体特征的类与具有部分特征的类之间的关系。需求陈述中若出现了“包含”、“组成”等字句，则往往意味着存在聚集关系。表示聚集的图形符号是在表示关联关系的直线末端紧挨着整体类的位置画一个菱形。

聚集关系具有传递性与反对称性。其中，传递性是指如果 A 包含 B，B 又包含 C，则 A 也将包含 C。反对称性是指如果 A 包含 B，则 B 不能包含 A。

聚集关系可以分为共享聚集与复合聚集两种形式。

如果在聚集关系中处于部分方的对象可同时参与多个处于整体方对象的构成，则该聚集称为共享聚集，这是一种较弱的聚集关系，整体对部分并没有什么专门的限制，它们分别独立存在，只不过部分可以合并起来，其图形符号是空心菱形。如图 6-17 中的装瓶的箱子与瓶子之间，就是这种共享聚集关系。

但是，如果部分类完全隶属于整体类，部分与整体共存，整体不存在了部分也会随之消失，则该聚集称为复合聚集，这是一种较强的聚集关系，也称组合关系，其图形符号是实心菱形。如图 6-18 中的窗口与窗口内控件之间的聚集关系。当在屏幕上打开一个窗口时，窗口内的文本框、列表框、按钮等控件会一同出现，而一旦关闭了窗口，则组成窗口的诸多控件也同时消失。因此，窗口和它的组成部分之间是复合聚集关系。

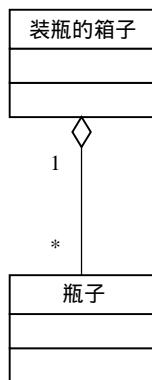


图 6-17 共享聚集关系

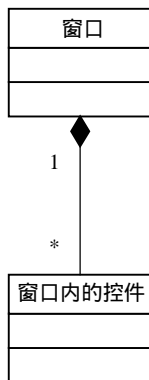


图 6-18 复合聚集关系

### (5) 泛化关系

类之间的泛化关系也就是前面介绍的类的继承关系，使用三角形图形符号表示，用于表明上级父类的属性、操作能够被下级子类继承。如图 6-2 中的“学校成员”、“教师”、“学生”、“新生”、“班主任”等之间存在的泛化关系。图 6-19 中的图书与纸质图书、电子图书之间也具有这种泛化关系。另外，用例模型中的扩展关系、使用关系则是两种特殊的专门针对业务类的泛化关系。

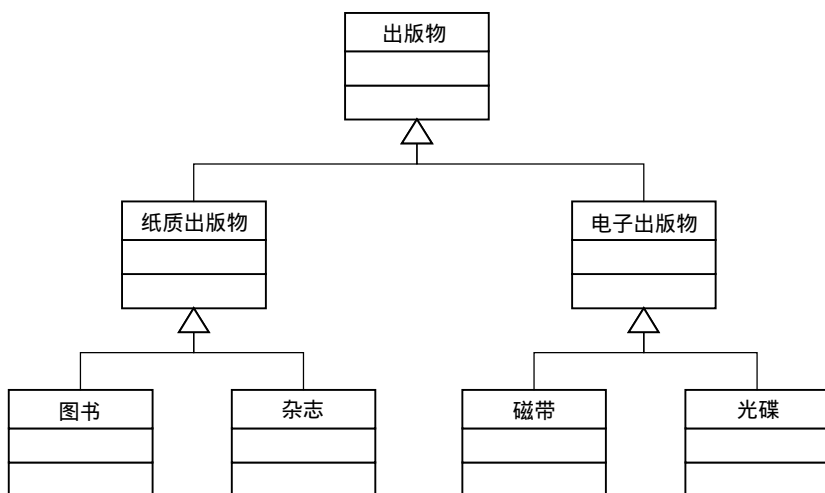


图 6-19 泛化关系

### (6) 依赖关系

依赖关系用于描述两个类之间存在的与依赖有关的语义上的连接关系。其中，一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，需要由独立元素提供服务，如果独立元素改变了，将影响依赖于它的模型元素。例如，一个类使用了另一个类的对象作为它的操作的参数，一个类使用了另一个类的对象作为它的数据成员，一个类工作依赖于另一个类向它发送消息等，这样的两个类之间就都存在着依赖关系。

在 UML 的类图中，依赖关系使用带箭头的虚线连接有依赖关系的两个类，其箭头指向独立的类。图 6-20 是“订单提交窗口”与“订单”之间存在的依赖关系，其中“订单”类是独立的，而“订单提交窗口”依赖于“订单”类。

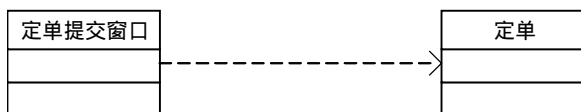


图 6-20 类之间的依赖关系

#### (7) 关系综合举例

上面逐个介绍了类之间存在的各种关系。实际问题中，各种关系往往会一起出现在同一个类图模型中。分析类图中存在的这些关系是类图建模的一项重要内容，它们将成为确定类之间消息通信的重要依据。

图 6-21 是“网上图书销售系统”的类图关系模型图。

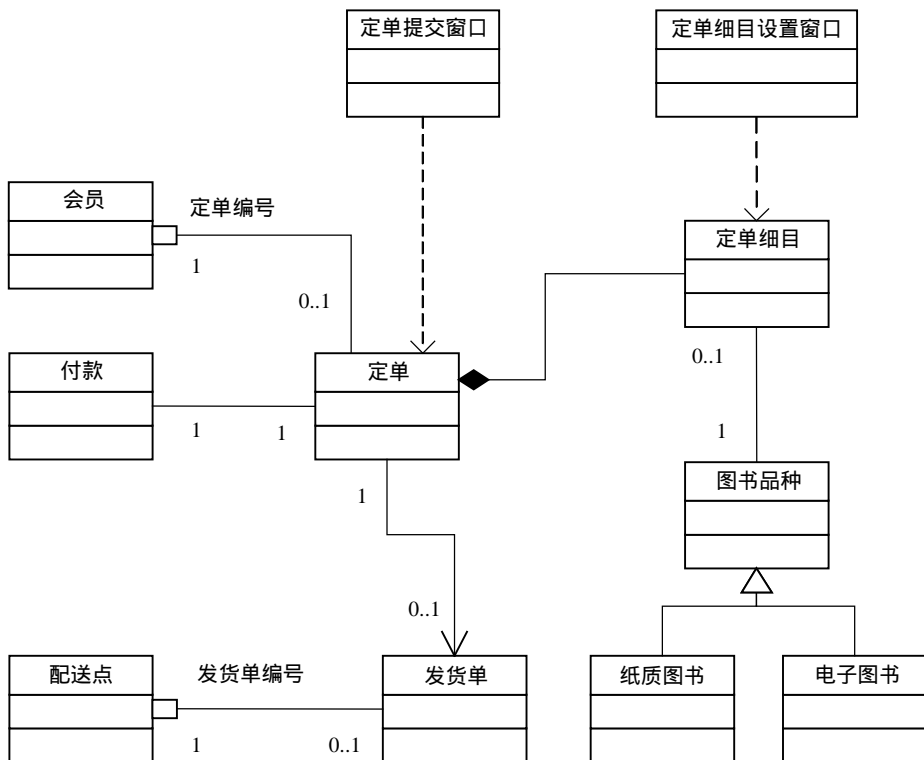


图 6-21 “网上图书销售系统”类图关系模型

### 6.2.4 序列图

序列图以用例图中的用例为描述单位，以类图中的类为对象依据，以活动图中的活动转换为行为依据，需要建立的是与时间有关的用例中对象之间的交互模型。

序列图中两个坐标轴，其中的纵坐标轴表示时间，横坐标轴表示不同的对象，由此而能够表现对象间消息传递的时间顺序。

序列图中的对象是类图或用例图中一个类的实例，用一个矩形框表示，框内标有对象名与对象所属的类。序列图中对象的命名格式是：[对象名]:类。

从表示对象的矩形框向下的垂直虚线是对象的“生命线”，用于表示一个对象在一段时间内的存在状态。

对象之间的消息通信用对象生命线之间的水平线表示，消息箭头形状表示消息的类型（如同步、异步），其中的消息可以用消息名和参数表来标识。

对象激活用对象生命线上的细长矩形框表示。当对象收到消息时，接收对象立即开始执行活动，即对象被激活了。

图 6-22 是“网上图书销售系统”中“产生定单”活动的序列图，其创建依据是前面的图 6-11 中活动图和图 6-21 中的类图。

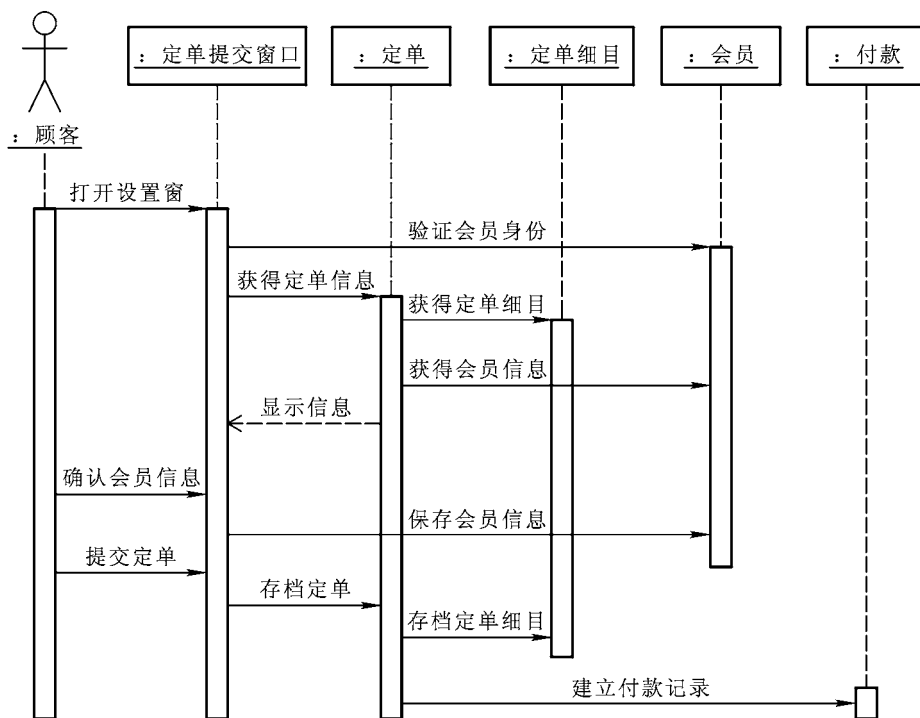


图 6-22 “网上图书销售系统”中产生定单活动的序列图

### 6.3 面向对象设计建模

面向对象设计建模需要把分析阶段的结果扩展成技术解决方案，需要建立的是软件系统的技术构造模型。因此，类图中的类由现实实体进化成为构造软件系统的类模块，有关类、对象、组件的建模都成为技术概念，并需要为软件系统的实现提供设计依据。

面向对象设计过程中的主要建模内容有：设计类图、协作图、状态图、组件图和部署图，包括基于设计类图、协作图和状态图的逻辑模型，也包括基于组件图和部署图的物理模型（其一般建模流程见 6.1.3 节中的图 6-5）。

### 6.3.1 设计类图

设计类图中需要考虑的类已经不只是实体类，还包括用于向外提供操作接口的边界类和用于实现内部协调的控制类。

设计类图中的类是构造系统的基本模块单位，因此，需要进行更加完整的面向设计的描述。一些遗漏的属性需要补充进来，需要通过分析时建立的序列图逐步完成对类的操作的描述，并需要针对属性、操作等进行符合设计要求的说明或注释。

#### 1. 表示属性

类中属性反映了对象的数据特征，其描述格式是：

[可见性] 属性名 [:类型] [=初始值]

表示属性的以上成分中，括号 [ ] 内的成分是可选内容。显然，表示属性时，只有属性名是必须的，其他都是可选的。其中属性的类型，可以是原始的数据类型，也可以是对另一个类的引用。

#### 2. 表示操作

类中操作反映了对象的行为特征，由发给对象的消息调用，其描述格式是：

[可见性] 操作名 [ ( 参数 ) ] [:返回类型]

如同属性，表示操作的以上成分中，括号 [ ] 内的成分是可选内容。显然，表示操作时，只有操作名是必须的，其他都是可选的。其中，操作中的参数可表示为：

[方向] 参数名:类型 [= 缺省值]

#### 3. 表示属性、操作的可见性

类中属性、操作具有可见性，其表明属性与操作在多大范围内能够被访问。

属性、操作的可见性通常分为三种：

(1) 公有的 (public)，用加号 ( + ) 表示，能够被所有具有访问接口的类访问。

(2) 私有的 (private)，用减号 ( - ) 表示，能够被它自己访问。

(3) 受保护的 (protected)，用井号 ( # ) 表示，能够被它自己以及它的下级子孙类访问。

通常情况下，类的属性大多被设置为私有的，以表明其内部数据是私有数据，外界不能直接干预。而类的操作则大多被设置为公共的，以表明其能够对外提供服务。

图 6-23 是 “网上图书销售系统” 的类图设计模型图。

### 6.3.2 协作图

协作图是类似于序列图的又一个对象行为模型，能够描述对象之间的交互关系，但序列图所表现的是对象交互的时间顺序，而协作图所表现的是对象交互时的链接关系和基于链接而产生的消息通信及其操作接口。

与序列图中一样，协作图中的对象是类图中类的实例，其图形符号、命名格式等也与序列图中一样。但协作图比序列图具有更多的设计元素，它可以通过连接对象的直线表示对象之间



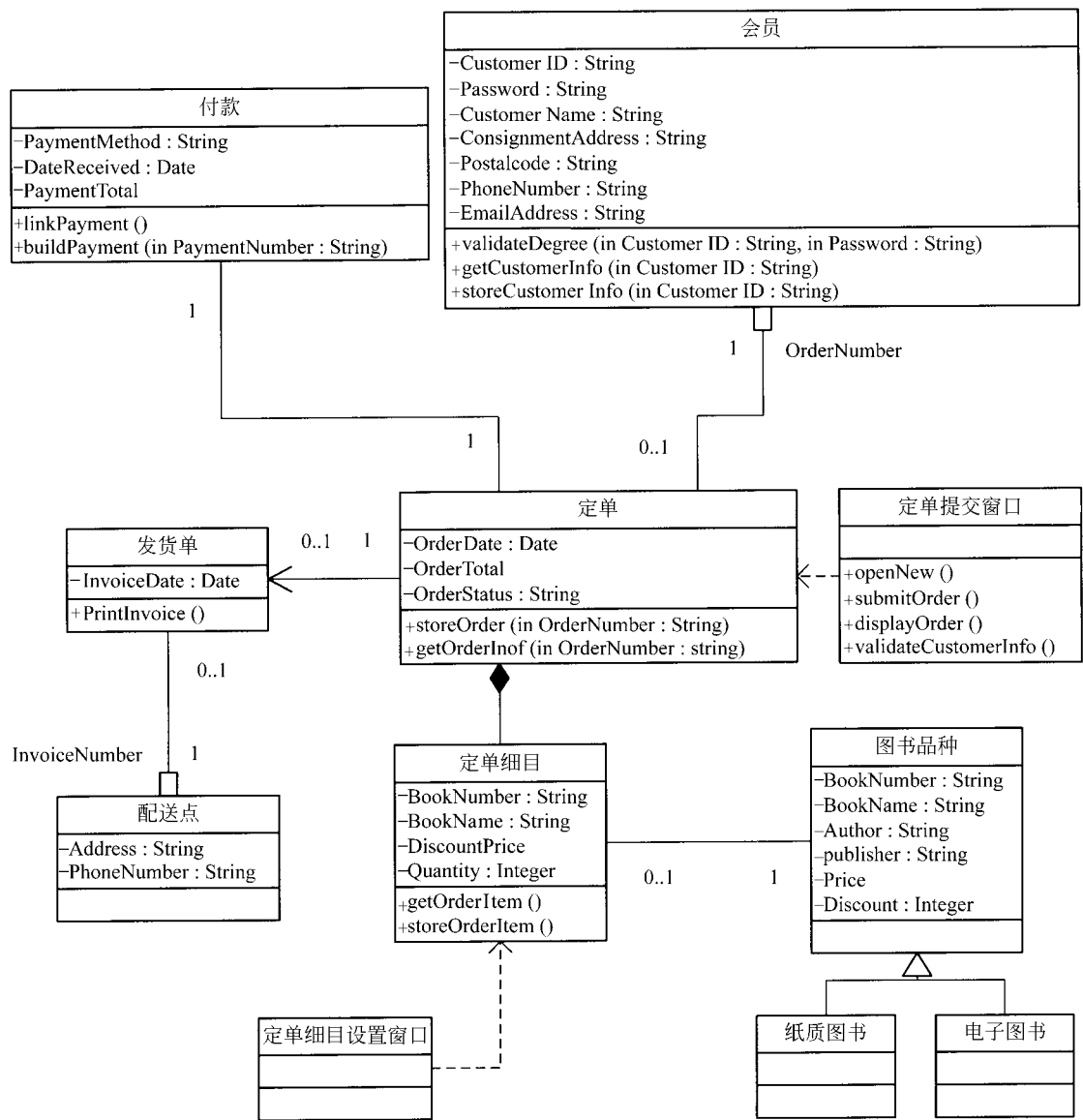


图 6-23 “网上图书销售系统”类图设计模型

的链接关系，而且链接线可以带有消息标签，用来表示对象之间的消息传递，为了说明消息在序列图中的时间顺序，可以在消息标签上加上序号。

协作图中对象之间的链接关系可以看作类图中类之间关联关系的实例，其消息则可对应到类图中类的操作。应该说，协作图与类图分别从动与静两个方面对对象与类进行描述。实际设计中，协作图一般与设计类图同步建模，并可依靠协作图而获得对类图更加合理的设计，例如，发现类图中操作定义的不合理性，或将类图中一些遗漏的操作补充进来。

图 6-24 是“网上图书销售系统”中“产生定单”活动的协作图。

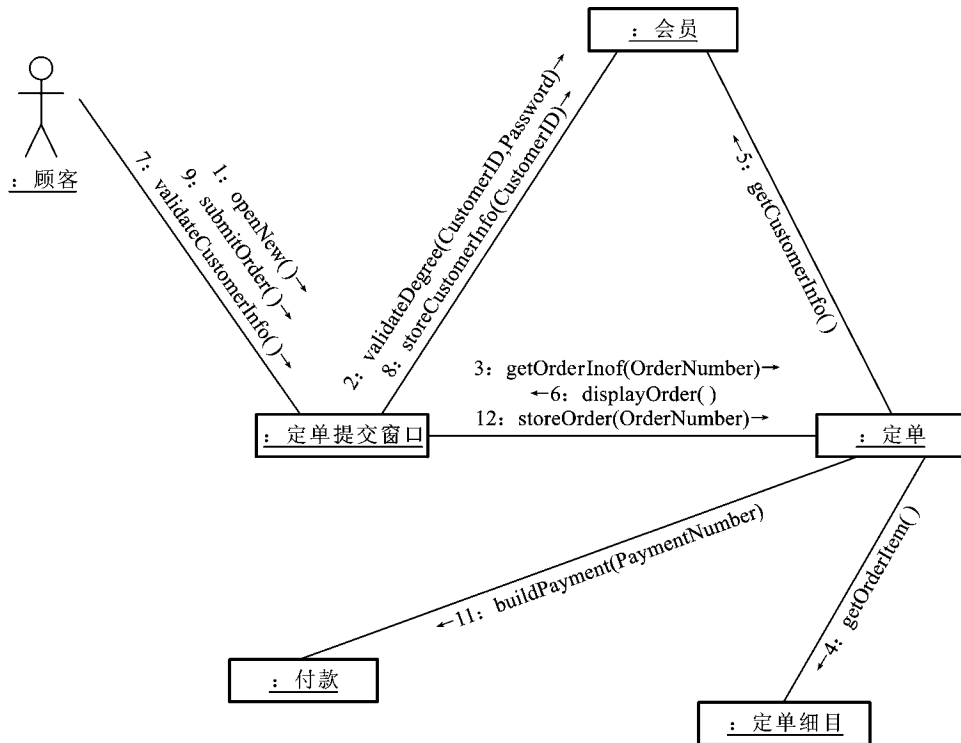


图 6-24 “网上图书销售系统”中“产生定单”活动的协作图

### 6.3.3 状态图

本书已在 4.4.4 节介绍了状态图。面向对象方法中主要使用状态图描述一个特定对象的所有可能的状态以及引起状态转换的事件。

应该说，状态图提供了对象在其生命期中可能出现的状态及其行为的描述。一个状态图包括一系列状态、事件以及状态之间的转移，其作用是能够为类图中每一个类进行动态行为说明，以获得对类的方法的细节描述的建模支持。

图 6-25 是“网上图书销售系统”中“定单”类的状态图。

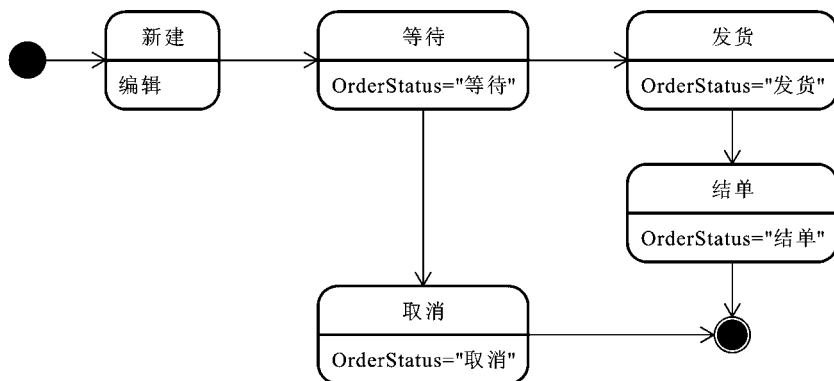


图 6-25 “网上图书销售系统”中“定单”类的状态图

### 6.3.4 构件图

构件是系统中一个具有清晰功能特征的物理单元，是软件系统逻辑架构中定义的概念和功能在物理架构中的实现，对应于组成软件系统的目标文件，涉及可执行程序文件、动态库文件、数据库文件、HTML 文件等。

可以将构件看作实现类，构件图的作用就是描述这些构件类及它们之间的关系。构件之间关系主要是依赖关系。

图 6-26 所示是“网上图书销售系统”的构件图。以其中的顾客服务为例：顾客可以通过 Web 页面访问系统，然而具体的顾客服务处理则依赖于顾客服务程序；若顾客需要查询图书信息，则需要通过图书信息查询程序获得服务；若建立购书定单，则需要通过定单处理程序获得服务；而在提交定单时，还需要建立付款记录，因此它还需要依赖于账务处理程序构件。

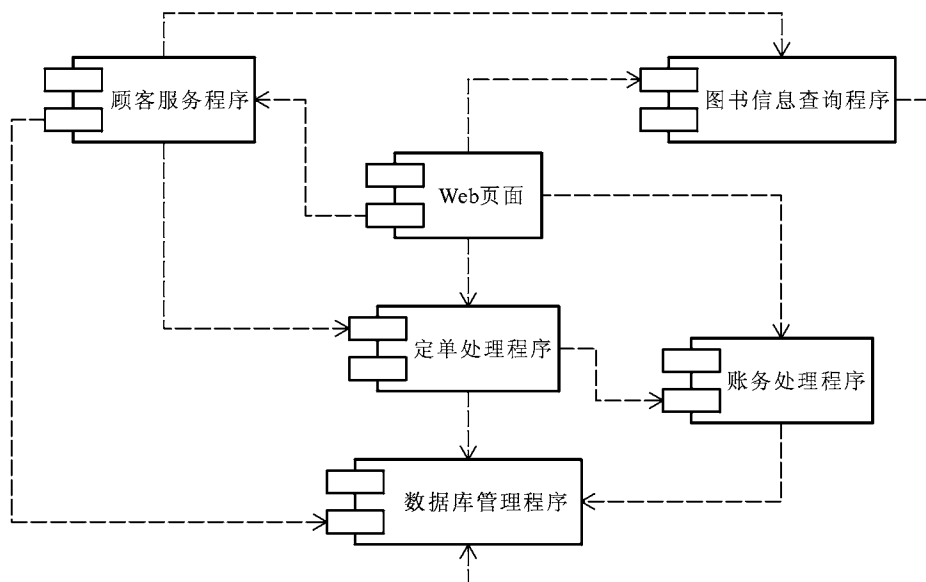


图 6-26 “网上图书销售系统”的构件图

在 UML 中，构件的图形符号是左边带有两个小矩形的大长方形，构件之间的依赖关系则用一条带箭头的虚线表示。可以为一个构件定义其他构件可见的接口，其图形符号是从代表构件的大矩形边框画出的—条线，线的另一端为一个—小空心圆，接口名写在空心圆附近。

设计时可以将类图中的类分配到需要创建的物理构件上去，以达到由逻辑设计往物理设计的过渡。当需要将逻辑类分配到物理构件去时，可以根据类在行为上的联系程度进行合理分配。

### 6.3.5 部署图

部署图用于描述系统运行时的物理架构，涉及物理节点、节点之间的连接关系以及部署到各个节点上的构件的实例等。

可以使用部署图说明硬件节点的拓扑结构、通信路径、节点上运行的软件构件等，可以将

节点、构件看作是分布式系统中分布单元，并可以使用部署图描述分布式系统的体系架构。如图 6-27 所示的“网上图书销售系统”的部署图，这是一个分布式应用系统，其涉及 Web 服务器、应用服务器和数据库服务器等几个节点。

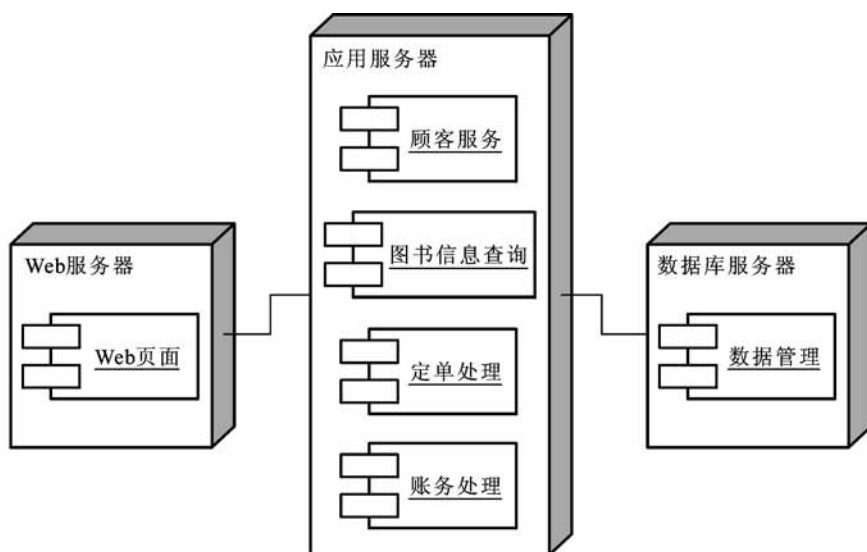


图 6-27 “网上图书销售系统”的部署图

## 小 结

### 1. 面向对象方法学

面向对象技术涉及面向对象分析（OOA）、面向对象设计（OOD）和面向对象编程实现（OOP）这三个方面的问题。

#### （1）基本概念

- 类：面向对象模块单位，作用是为创建对象实例提供模板。其具有数据与行为这两个方面的特征，并需要通过属性、操作和方法进行描述。
- 属性、操作与方法：类具有数据与行为这两个方面的特征，并需要通过属性、操作和方法进行描述。
- 类的继承性：指上级父类能够把自己的属性、操作传递给下级子类。
- 类的多态性：子类对象可以像父类对象那样使用，它们可以共享一个操作名，然而却有不同实现方法。
- 对象：对象是类模块实例化的结果。
- 消息：指对象之间的通信。

#### （2）优越性

- 跟现实世界更加接近
- 可使软件系统结构更加稳定

- 软件具有更好的可重用性
- 软件更加便于维护与扩充

## 2. 面向对象分析建模

面向对象分析建模需要建立的是软件系统的用户领域模型，需要从系统业务流程、组织结构和行为过程等几个方面对系统进行分析。

### (1) 用例图

用例图涉及参与者、用例等元素，用于描述用户与系统之间的交互关系，说明系统所具有的业务能力和业务流程，能够方便开发者理解用户领域的专有术语和业务内容。

### (2) 活动图

活动图是一种行为模型，主要用于描述用例图中用例的内部活动状态与活动转换过程，以获得对用例的交互行为与工作流程的细节说明。涉及活动状态、活动转换等元素。

### (3) 分析类图

建立类图的概念模型，描述体现现实世界中数据构造的实体类及其它它们之间的关系。

### (4) 序列图

以用例图中的用例为描述单位，以类图中的类为对象依据，以活动图中的活动转换为行为依据，建立与时间顺序有关的用例中对象之间的交互模型。

## 3. 面向对象设计建模

面向对象设计建模需要把分析阶段的结果扩展成技术解决方案，需要建立的是软件系统的技术构造模型。

### (1) 设计类图

设计类图中的类是构造系统的基本模块单位，需要在分析类图基础上进行更加完整的面向设计的描述。除了实体类，设计类图中还需要考虑用于向外提供操作接口的边界类和用于实现内部协调的控制类。

### (2) 协作图

描述对象交互时的链接关系和基于链接而产生的消息通信及其操作接口。

### (3) 状态图

描述一个特定对象的所有可能的状态以及引起状态转换的事件。

### (4) 构件图

描述组成系统的物理构件及其它它们之间的关系。构件之间关系主要是依赖关系。

### (5) 部署图

描述系统运行时的物理架构，涉及物理节点、节点之间的连接关系以及部署到各个节点上的构件的实例等。

## 习 题

1. 试说明面向对象中的类模块与传统结构化方法中的模块的区别。
2. 试描述类的继承性与多态性的作用。
3. 与传统结构化方法比较，面向对象方法具有哪些方面的优越性？

4. 试描述 UML 建模过程。

5. 第 4 章的 4.4.2 节曾使用数据流图分析了一个“工资管理系统”，其涉及职工清单、档案工资、业绩工资、工资报表等诸多数据，涉及人事处、财务处和员工所在工作部门等多个业务部门。现要求使用面向对象方法对该系统进行分析与设计，试建立有关该软件问题的用例图、活动图、类图、序列图和协作图。

6. 某“仓库管理系统”用户需求陈述如下：

(1) 仓库管理系统将被计划部门、仓库管理部门、采购部门、销售部门的相关工作人员使用。其中，计划部门需要制定商品计划。仓库管理部门需要进行商品入库、出库、报损等日常事务管理。采购部门需要查询商品库存情况、获取商品定货计划表。销售部门也需要查询商品库存情况和提出商品定货请求。

(2) 由于不同部门有不同的任务，因此系统需要提供针对部门权限管理机制和针对工作人员的登录注册机制。系统将通过一位系统管理员进行部门授权与工作人员注册管理。其中使用仓库管理系统的工作人员需要有惟一的个人身份标识，它既是工作人员登录系统时的身份验证依据，也是工作人员在进行商品操作时的经手人标记。

(3) 仓库中的商品需要以品种为单位进行管理，所有商品都要由计划部门按品种进行登记，涉及商品编码、名称、类别、库存下限值等数据。

(4) 仓库商品涉及入库、出库、报损这三种事务处理，商品的任何流通都需要以流水方式记录到商品流通表中，并对商品库存量进行更新。当商品出库、报损时，必须考虑到该商品的当前库存量是否能够满足操作需要。出库、报损后，若商品库存量低于库存下限值，将会自动产生定货请求。

(5) 仓库管理系统需要自动在月底对商品流通数据进行盘查，需要按月打印商品流通分类汇总报表。

试根据上述需求陈述建立起有关该软件问题的用例图、活动图、类图、序列图和协作图。

## 第7章 用户界面设计

用户界面设计所面对的是用户的应用接口。对于交互式系统而言，用户界面设计是否成功将直接影响着软件系统的质量，并最终影响着用户对软件系统的满意程度。

实际上，当用户对软件系统进行评价时，系统的用户界面将会备受关注。也许软件系统在技术上非常先进，但如果用户发现其很难操作，那么他们就很难接受它。可以想象到的情形是，一个不便于操作的界面可能会使用户的操作经常出现错误，并会使用户产生对软件系统的厌烦心理，由此直接影响软件的应用价值。

更加全面地说，图形用户界面的设计要求以用户为中心，应该使用用户术语实现与用户的交互。界面还应该具有逻辑性和一致性，应该具备帮助用户使用系统和从错误当中恢复的功能。

### 7.1 用户界面设计过程

#### 1. 图形用户界面（GUI）的特点

早期的用户界面大多是基于命令方式的，界面外观简单，主要是技术问题，一般由软件设计人员独立完成。然而，现今的用户界面则大多是图形用户界面（GUI），并具有以下几方面的特点：

（1）比较容易学习和使用，没有计算机基础的用户经过短期培训就能学会使用这种界面。

（2）用户可利用多屏幕（窗口）与系统进行交互，并可通过任务窗方便地由一个任务转换到另一个任务。

（3）可以实现快速、全屏的交互，能很快在屏幕上的任何地方进行操作。

显然，图形用户界面设计已经不是设计人员能够独立解决的软件技术问题了，它还涉及图形学、美学、行为学、心理学和社会学等其他学科方面的问题，因此需要考虑邀请图形设计人员、系统分析人员、系统设计人员、程序员、用户应用领域方面的专家和社会行为学方面的专家，以及最终用户的共同参与。

#### 2. 基于原型的用户界面设计

在为应用程序设计界面时，一次就设计出非常完美的界面的情况非常少见。这意味着，用户界面设计是一个迭代的过程，需要进行多次反复而使界面设计逐步趋于完善。

图 7-1 是用户界面设计的基本过程图，其包括以下三个步骤：

（1）建立界面需求规格模型。

（2）以界面需求模型为依据创建界面原型。

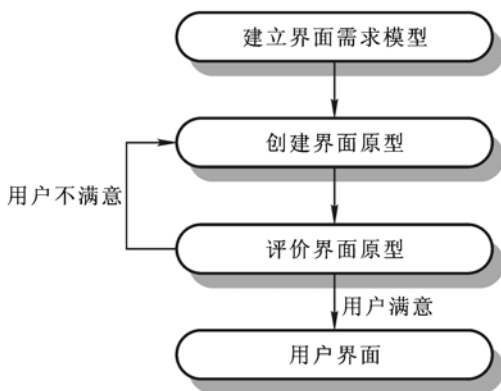


图 7-1 用户界面设计过程

### (3) 评价界面原型。

以在上述三个步骤中界面原型创建进行界面设计迭代。

需要注意的是,在界面设计过程中必须邀请用户参与,而且用户参与界面设计的过程越早,则在界面设计问题上所花费的精力越少,创建的界面会越具有可用性。

## 7.2 界面设计中需要考虑的因素

在设计用户界面时,需要随时想到用户。例如,如何才能使用户在没有专门指导的情况下,自己就能发现应用程序的各种功能?当有错误发生时,应用程序具有哪些排错措施,能够让用户从错误陷阱中跳出来?界面设计是否能够通过一种艺术美来使用户感觉舒适?当用户需要帮助时,它能够通过哪些方式获得帮助?等等。

实际上,用户界面设计时将会受诸多来源于用户的因素的影响,并主要体现在以下几个方面。

### 1. 用户工作环境与工作习惯

用户界面设计需要解决的是用户与软件系统的交互问题,因此,用户在使用软件时需要完成的工作将直接决定着用户界面的基本格局。

对此,需要考虑以下方面的用户问题:

(1) 用户的工作环境。用户界面中的元素及其操作应该与用户的工作环境尽量贴近。例如某电路控制系统的界面,其界面元素就应该是控制开关、控制线路,相关操作则应该是通电、断电、合闸、拉闸等。

(2) 用户的工作习惯。界面设计还应该考虑到用户工作中已经形成的习惯。例如不要与用户原有的业务流程发生冲突,使用用户熟悉的领域术语等。

### 2. 用户操作定势

用户操作定势是指用户在使用系统时形成的一种思维定势。如果某个操作引起某种变化,那么他就有理由相信另一个相似的操作所能够带来的变化应该具有相似的结果,如果产生了完全不同的结果,用户就会吃惊,并会感到困惑。

为了满足用户操作上形成的思维定势,界面设计者必须尽量使类似的操作能够具有类似的效果。

### 3. 界面一致性

界面一致性是指系统界面具有一致的风格。例如,系统中的各个界面具有相同的命令格式,具有相似菜单与工具栏构造,相似的操作具有非常接近的触发方式等。

一致的界面可以在软件系统中创造一种和谐,任何东西看上去都那么协调,并可以减少用户的学习时间,使用户在一个界面上学习到的操作,可以应用到其他的界面上去。而如果界面缺乏一致性,则很可能引起混乱,使软件显得没有条理。例如用于编辑文本的编辑器,如果一个地方使用了白色背景,除非有很好的理由,否则不应该在别的地方又使用灰色。

为了使得界面具有一致性,在进行界面设计时需要制定统一的设计规则或约定,否则将可能带来混乱。例如,键盘命令“Ctrl+C”,在系统的一个界面上被设置为“复制”的快捷键,而在另一个界面上则被设置为“剪切”。可想而知,这种不一致的设置必会带来操作上的混乱。



#### 4. 界面动作感

动作感是一种感觉,它能够带来操作线索。例如录音机上的按钮,一看到它的外形就可以想到它所带来的操作及用途。

用户界面设计也存在动作感。例如命令按钮往往被设置为三维立体效果,这可以使得这些按钮看上去像是可以被按下去的东西。相比之下,如果命令按钮被设计成平面边框,它的动作感就会下降。文本框也有动作感,用户可能习惯性地认为,可编辑的文本框是带有边框和白色背景的框。

很显然,界面设计中控件的动作感是非常有用的额外提示信息,可以使界面更容易被用户理解,界面设计者应该充分利用。

#### 5. 界面信息反馈

界面信息反馈是界面应该向用户提供有关界面操作方面的信息,使得用户知道当前将要进行的操作将会产生什么结果,执行进度如何。因此,界面设计者应该考虑如何在系统工作过程中为每个用户建立必要的信息提示。

在大多数情况下,一个沙漏或一个等待指示器就提供了足够的反馈来显示程序正处于执行状态。但对那些可能需要较长时间才能完成的操作,则需要一个更生动的反馈形式,例如进度指示器。

#### 6. 个性化

个性化是指用户对界面外观的个性化设置。如设置界面的背景色、字体,界面中子窗口的布局等,用户可以将这些设置保存起来,以作为他的个人偏爱,当下一次程序激活时,界面将出现他所需要的界面外观。

#### 7. 容错性

容错性是指用户界面应该以一种宽容的态度允许用户进行实验和出错,例如能够自动修正用户在文本框中输入的不正确的数据格式或给出错误提示。容错性也意味着允许用户进行操作试探,例如允许用户选择错误的执行路径并在需要的时候能“转回”到开始点。

容错性还隐含着可以进行多级取消操作,这一点容易说但不容易做,尤其是在多用户数据库的应用中。例如从银行账号中进行转出资金操作,这时对数据库的操作就无法直接取消,而必须通过在另一次事务中将资金存入账户的方式来改正这个问题。

#### 8. 审美性与可用性

审美性是界面视觉上的吸引力,而可用性则指界面在使用上是否方便、简单、有效以及可靠性和效率。它们是两个不同方面的问题,但相互关联,都将影响用户对界面的满意度。

在考虑界面的美学特性的时候,需要考虑的问题包括人眼睛的凝视和移动、颜色的使用、平衡和对称的感觉、元素的排比和间隔、比例的感觉、相关元素的分组等。

可以说,在用户界面中使用空白空间有助于突出元素和改善可用性。试想一幅人物画,如果人物将整个画面充满了的话,则人物反而会突出不起来。界面设计也是如此,若元素将整个界面充满,则元素就难以突出显现了。在一个窗体上放置太多的元素,往往会导致界面杂乱无章,因此在设计界面时,应该考虑留有空白空间。

界面中元素之间一致的间隔以及垂直与水平方向元素的对齐也可以使设计更具可用性。就像杂志中的文本那样,排列得行列整齐、行距一致,这样的界面会显得更加清晰。

在界面设计中，各元素的重要性并不完全一样。仔细地思考每个元素的作用，以确保越是重要的元素越要尽早地显现给用户。重要的或者频繁访问的元素应当放在显著的位置，而不太重要的元素就应当降级到不太显著的位置。

界面设计中还需要考虑元素的位置。人的一般阅读习惯是从上到下、从左到右，对于计算机屏幕也如此。大多数用户的眼睛会首先注视屏幕的左上部位，所以最重要的或必须最先操作的元素一般也就应当放在屏幕的左上位置，而最不重要或需要最后操作的元素则一般放在屏幕的右下位置。

界面的审美和可用性的问题需要界面设计者如同一个艺术家一样的工作。在这里应该记住原则是“简单即是美”。那么，界面要达到什么设计程度才能称得上简单呢？最有效的检验方法就是在软件的应用中观察用户的反应。如果有代表性的用户没有联机帮助就不能独立完成想要完成的任务，那么就有必要重新考虑界面设计，需要对界面做进一步的简化。为了简化复杂界面，最有效的方法是对界面按任务进行分解，由此可以使界面上的信息逐步地显现出来。

为了提高界面的审美性与可用性，界面设计中还需要考虑如何使用颜色，以增加对人的视觉影响。但是颜色不能滥用，目前的显示器一般都具有了显示上千万种颜色的能力，面对这许多的颜色，如果在设计时不做仔细地考虑，颜色也会像界面中的其他元素一样出现许多问题。应该说每个人对颜色的喜好有很大的不同，同一种颜色对于不同的用户，可能会引发不同的情感。因此在界面设计时最好是保守传统，采用一些柔和的、更中性化的颜色，如灰色。少量明亮色彩可以有效地吸引人们对重要区域的注意，但应当限制应用程序所用颜色的种类，而且色调也应该保持一致。

## 7.3 界面类型

在基于图形界面的应用系统中，用户界面一般由若干个窗体组成，窗体类型可以包括 SDI 单窗体、MDI 主窗体、MDI 子窗体、辅助窗体（对话框）等。假如系统是基于 Web 的应用开发，则用户界面中就需要用到 Web 页面。

### 7.3.1 单窗体界面（SDI）

SDI 界面的特点是应用程序一次只能打开一个独立窗体。如 Microsoft Windows 中的“写字板”，它使用的就是 SDI 界面，如图 7-2 所示，每次只能打开一个文档。

SDI 界面可以作为一个独立主窗体使用，通常包含标题条、菜单条、工具条、状态条以及窗体工作区等几个区域。其中的菜单条通常采用层次结构用于组织系统的功能，并通过菜单条中的菜单项提供对系统中各项功能的调用命令。

工具条则包含菜单条中最经常使用的菜单项的行为图标，它重复这些菜单项，提供一个执行常用功能的快速方式。

当软件问题比较复杂时，SDI 窗体的工作区可以被组织成几个窗格，各窗格中允许查看和操作不同但相关的信息内容。如图 7-3 所示的 Windows Explorer，它是一个用于文件搜索的独立窗体，为了方便对文件的搜索，窗体被分成两个窗格，其中左半部分是一个树型路径视图，右半部分用于显示左半部分对应路径下所包含的内容。

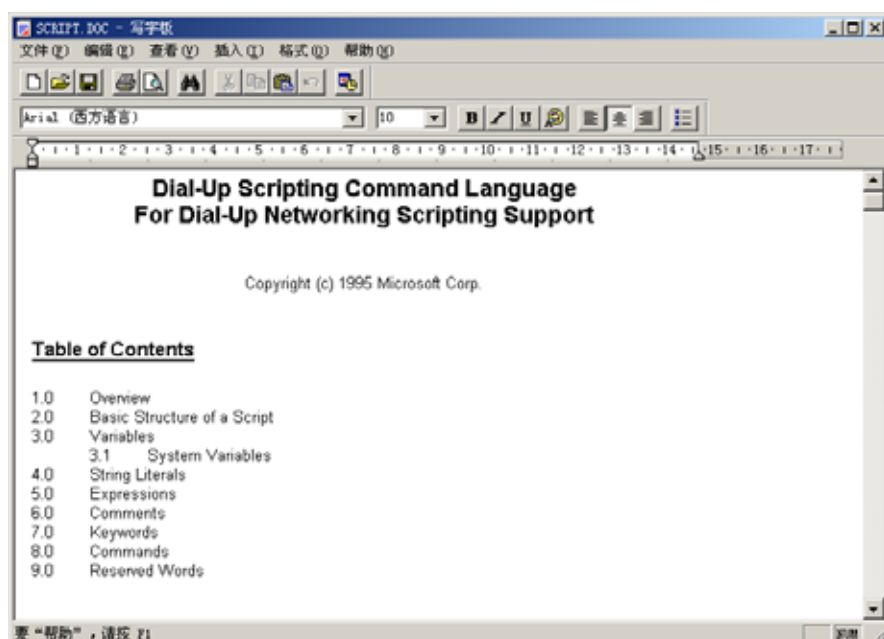


图 7-2 SDI 窗体

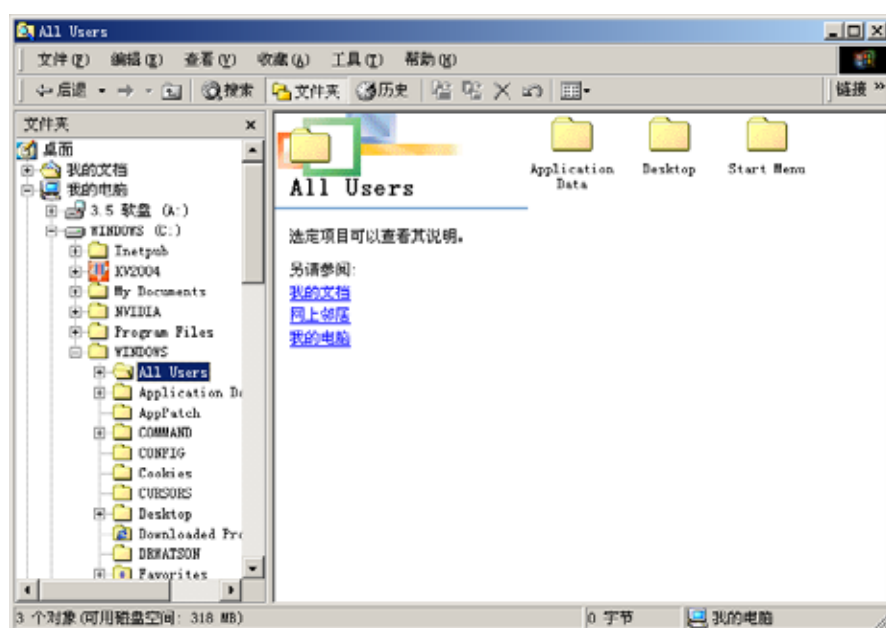


图 7-3 被组织成两个窗格的窗体

### 7.3.2 多窗体界面 (MDI)

当应用程序需要处理更加复杂的问题时，例如需要同时处理多个文档时，则需要同时打开多个文档，这时就要用到 MDI 界面。例如 Microsoft Excel，由于有可能要同时操作两个以

上的 Excel 文档，它所采用的就是 MDI 界面。

一个 MDI 界面由一个 MDI 主窗体和多个 MDI 子窗体组成。其中 MDI 主窗体如同容器用来装载 MDI 子窗体，而 MDI 子窗体则被限制于 MDI 主窗体之内，不能独立存在。因此诸多公共操作都被放置在 MDI 主窗体上，例如菜单条、工具条、状态条等。

图 7-4 中的“图书馆管理系统”就是 MDI 界面，其 MDI 主窗体中包含着“读者借书还书”、“图书信息查询”两个 MDI 子窗体。

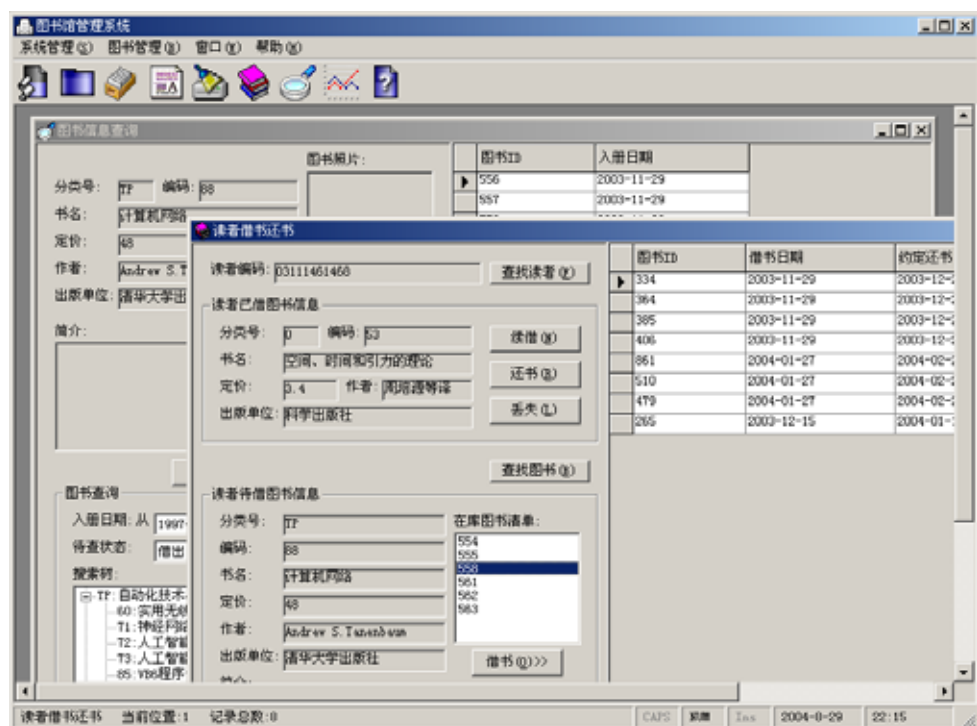


图 7-4 MDI 界面

### 7.3.3 辅助窗体

辅助窗体通常也叫做对话框，它是对主窗体的补充，用于扩展主窗体的功能，能够通过主窗体上的菜单命令的调用被打开，运行时置于主窗体的前面。

相对于主窗体，辅助窗体通常是模态的，使得用户在与任何其他的应用窗体进行交互之前，必须先关闭辅助窗体。

辅助窗体的种类主要有以下几种。

#### (1) 登录窗

登录窗通常工作在主窗体的前面，用于验证登录系统的用户的身份是否合法。

登录窗一般由文本框、命令按钮等控件组成，需要输入用户名、密码等信息。为了防止密码被其他的人有意或无意看到，密码输入框通常都做了掩码设置。图 7-5 即是一个用户登录窗。

## （2）消息窗

消息窗用于向用户显示程序运行时的一些辅助信息，它们可以是一个警告信息、一条解释或一个异常条件等。消息窗中通常通过命令按钮给用户提供一个或多个回复选择。例如图 7-6 所示的消息窗，这条消息用于请求用户对删除记录操作进行确认。

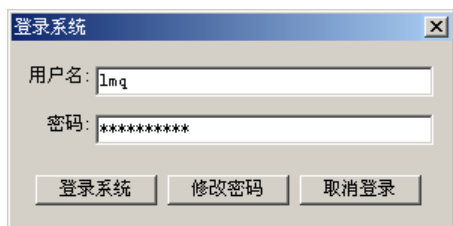


图 7-5 登录窗



图 7-6 消息窗

## （3）设置窗

设置窗用于对系统提供参数设置对话功能。例如存储路径设置、计算标准设置、报表格式设置、商品类别设置、用户组群设置等。当需要通过一个设置窗进行多项参数设置时，可以在设置窗中使用标签条进行分组。例如图 7-7 所示的 Microsoft Word 中的“选项”设置窗。

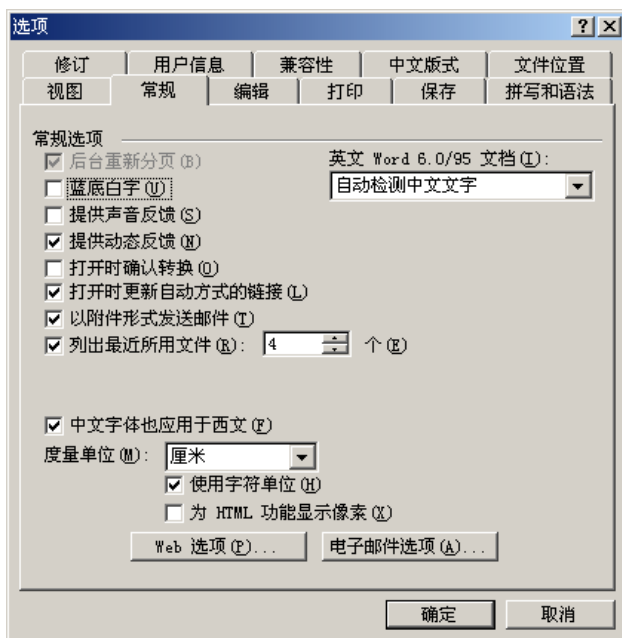


图 7-7 设置窗

### 7.3.4 Web 页面

现今的许多系统为了互联网通信的需要，或考虑到客户端维护的便利，往往采用基于 Web 的 B/S 结构。在这种情况下，如果系统中的某个 Web 页面被作为一个 Web 应用的进入点，则它

可以作为一个特殊的主窗体看待。

在 Web 应用中,用户事件一般通过行为按钮和超链接来编程。图 7-8 是“首信易支付”Web 首页图。



图 7-8 Web 页面

## 7.4 界面功能特征

在进行用户界面设计时,需要考虑界面的功能问题。大体上说来,用户界面的功能主要体现在三个方面:(1)用户与界面之间的交互;(2)系统信息在界面上的表示;(3)系统对新用户的学习指导。

### 7.4.1 用户交互

用户交互是指用户与计算机系统之间的信息交流。在早期的计算机上,用户交互的惟一方式是使用专用语言与机器进行交流,实际上这种早期方法只有专家才能使用。但在今天,用户有了更加方便、有效的与系统进行交流的方式,具体说来有以下几种:

(1) 直接操纵。用户在屏幕上直接与对象进行交互。例如用户要删除一个文件,就把它拖到回收站中。

(2) 菜单选择。用户从一系列可选的菜单命令中选择出一个命令。通常情况是另一个屏幕对象同时被选中,命令作用于这个对象。用这种方法删除一个文件时,用户先选定这个文件,然后选定删除命令。

(3) 表格填写。用户填写表格的空白栏。有些空白栏可能有相关菜单，表格上可能有操作“按钮”，按下时就会开启其他的操作。用基于表格的界面删除文件是一种人工操作，先要填入文件名，然后“按”删除按钮。

(4) 命令语言。用户把特定的指令和相关参数发送出去，指示系统该做什么。如果删除一个文件，用户发出删除指令将文件名作为参数。

(5) 自然语言。用户用自然语言发出指令。因此如要删除一个文件，用户要键入“删除名为××的文件”。

应该说上述每一种类型的交互都存在优点和缺点，分别适用于不同类型的应用和用户。这些交互类型也可以混合使用，几种不同的类型可以用于相同的应用。表7-1列出了这些类型的主要优点和缺点及其所适用的应用类型。

表7-1 各种用户交互方式比较

交互方式	主要优点	主要缺点
直接操纵	快速、直观，容易学习	较难实现，只适对象化操作
菜单选择	可避免用户出现操作错误，只需要很少的键盘输入	对于有经验的用户来说，其操作速度太慢
表格填写	简单的数据入口、容易学习	占据了太多的屏幕空间
命令语言	功能强大、灵活	较难学习
自然语言	适合偶然用户、容易扩展	需要太多的键盘操作

## 7.4.2 信息表示

所有的交互式系统都要提供给用户某种方式的信息表示。信息可以采用文本形式表示，也可以采用图形形式表示。在许多界面设计中，采用了把用于信息表示的程序与信息本身相分离的方法，如图7-9所示。应该说这是一种比较好的系统设计方法，有利于同一个信息以不同的方式表示。

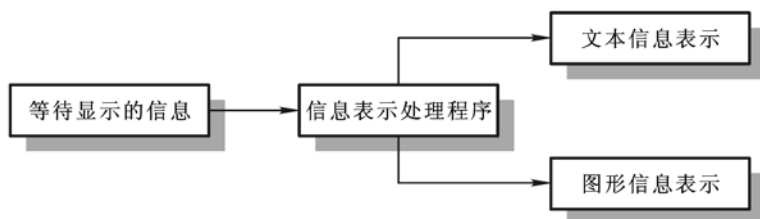


图7-9 信息表示

实际上，不同的用户由于应用目的、知识程度的不同，会有不同的信息表示需求。这使得界面设计者在决定如何表达信息的时候，必须考虑以下方面的用户问题：

- (1) 用户对不同数据之间的关系感兴趣吗？
- (2) 信息表示刷新速度如何？数据的变化需要马上显示给用户吗？
- (3) 用户必须根据数据的改变执行某种操作吗？
- (4) 用户需要通过直接操作界面对象而与显示的信息进行交互吗？

(5) 要显示的信息是文本形式的还是数字形式的？信息项的相对值重要吗？

应该说，不同的信息可能需要不同的表示形式。一般说来，如果需要精确的数字信息并且信息变更相对较慢，信息应该以文本形式表示。如果数据变更的较快或者需要表示数据之间的关系，则需要使用动态图形形式表示。

当需要表示物理实体时，还可能需要用可视化方法表示信息。例如使用一组可视节点表示电话网络状态，使用三维技术表示分子模型等。

在表示信息时，往往需要用到各种不同的颜色。应该说颜色能够改善用户界面，但同时颜色也容易被误用。一般而言，用户界面颜色的使用应该保守一些为好，并需要注意以下几个方面的问题：

(1) 限制使用颜色的数量。在一个窗口中不应使用四种以上的颜色，在一个系统中使用的颜色不应超过七种。

(2) 尽量使用中性颜色，如灰色、白色、黑色，原因是太明亮的颜色会增加视觉疲劳。

(3) 尽量不使用颜色表达特定的含义。原因是有些用户可能有色盲，他们可能会误解其中的含义。另外不同的职业对于颜色有不同的理解。

(4) 注意色彩搭配。由于眼睛的生理特性，人们不能同时凝视红色和蓝色。蓝色背景上的红色显示会使眼睛疲劳。另外的一些色彩组合也可能使视觉紊乱或难以读取。

### 7.4.3 用户联机支持

用户联机支持也就是系统给用户提供的应用指导，包括系统运行时的错误消息提示和联机帮助系统两个方面的内容。

#### 1. 错误消息

错误消息用于提示系统运行中产生的错误，这些错误可能来自于用户的错误操作，也可能来自于系统自身。错误消息提示的作用是指导用户在遇到错误时需要如何应对，因此它应该是有礼貌的、简洁的、一致的和建设性的，而不应该有侮辱性的成分，也不应该伴有响声或者其他的噪声，以免使用户难堪。可能的话，应该给出信息以提示如何改正错误，错误信息应该链接到上下文相关的在线帮助系统上。

#### 2. 联机帮助

当用户使用系统初期，或在操作中出现困惑时，或读不懂面前出现的错误信息的时候，可以求助于联机帮助系统。不同的系统可能需要不同类型的联机帮助。通常情况下，联机帮助系统中的文档应当组织成与软件系统功能组织保持一致的树形结构，以利于用户的学习。图 7-10 是 Microsoft Word 的联机帮助系统。

联机帮助系统中文档内容的编写应该有应用领域方面专家的参与，以保证帮助文档能够按照用户的术语风格进行说明。帮助文本本身、文本编排和风格需要仔细设计。帮助文字说明应该简要、通俗，并提供链接功能，以利于用户的阅读。帮助文档中的画面不只是系统工作时的屏幕抓图，还需要提供足够的注释说明，以利于用户轻松看懂画面。

联机帮助系统应该提供多种不同信息检索方式，并具有前后搜索功能以利于用户快速获取帮助。实际上帮助系统可以依靠专门工具创建。



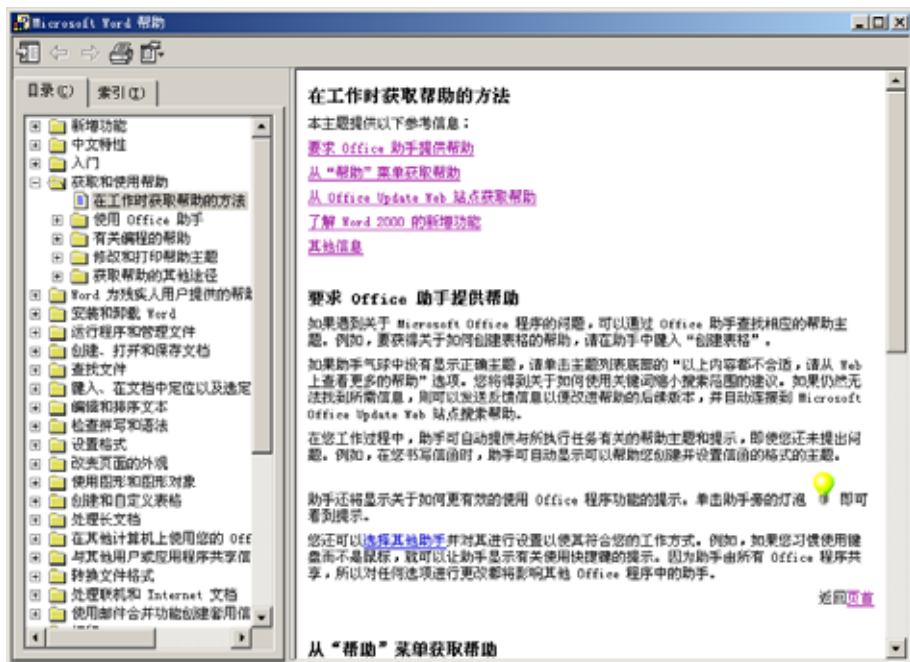


图 7-10 联机帮助系统

## 7.5 界面导航设计

界面导航所指的是如何由一个界面转换到另一个界面。可以使用活动图来描述界面之间的转换关系，其中活动图中的每一个活动状态可用来表示系统中的每一个界面。图 7-11 是某图书馆“图书借阅管理系统”的窗体导航图。

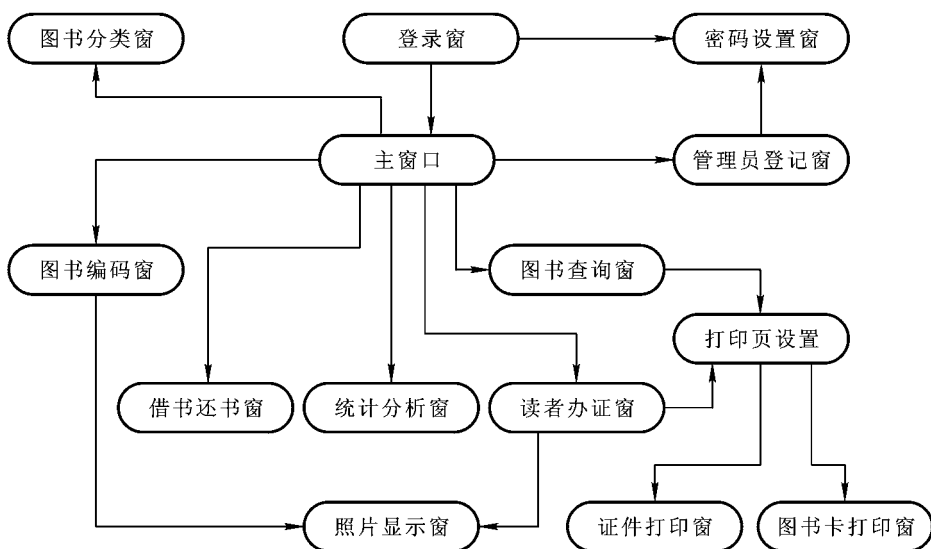


图 7-11 “图书借阅管理系统”窗体导航图

## 小 结

### 1. 图形用户界面 (GUI) 所具有的特点

(1) 比较容易学习和使用。

(2) 用户可利用多屏幕 (窗口) 与系统进行交互, 并可通过任务窗方便地由一个任务转换到另一个任务。

(3) 可以实现快速、全屏的交互, 能很快在屏幕上的任何地方进行操作。

图形用户界面设计已不是设计人员能够独立解决的了, 需要邀请图形设计人员、系统分析人员、系统设计人员、程序员、用户应用领域方面的专家和社会行为学方面的专家以及最终用户的共同参入。

### 2. 基于原型的用户界面设计

用户界面设计是一个迭代的过程, 其基本过程包括三个步骤:

(1) 建立界面需求规格模型。

(2) 以界面需求模型为依据创建界面原型。

(3) 评价界面原型。

### 3. 界面设计中需要考虑的因素

用户界面设计将会受诸多用户因素的影响, 并主要体现在以下几个方面:

(1) 用户工作环境与工作习惯。

(2) 用户操作定势。

(3) 界面一致性。

(4) 界面动作感。

(5) 界面信息反馈。

(6) 个性化。

(7) 容错性。

(8) 审美性与可用性。

### 4. 界面类型

在基于图形界面的应用系统中, 用户界面一般由若干个窗体组成, 其窗体类型包括:

(1) 单窗体界面 (SDI)。其特点是应用程序一次只能打开一个独立窗体。

(2) 多窗体界面 (MDI)。由一个 MDI 主窗体和多个 MDI 子窗体组成。其中 MDI 主窗体如同容器用来装载 MDI 子窗体, 而 MDI 子窗体则被限制于 MDI 主窗体之内, 不能独立存在。诸多公共操作都被放置在 MDI 主窗体上。

(3) 辅助窗体。通常也叫做对话框, 它是对主窗体的补充, 用于扩展主窗体的功能。辅助窗体的种类主要有: 登录窗、消息窗、设置窗等。

(4) Web 页面。当采用到基于 Web 的 B/S 结构时, 系统中的某个 Web 页面可能会被作为 Web 应用的进入点, 则它可以作为一个特殊的主窗体看待。

### 5. 界面功能特征

在进行用户界面设计时, 需要考虑界面的功能问题。大体上说来, 用户界面的功能主要体

现在以下方面：

(1) 用户交互。指用户与计算机系统之间的信息交流。

(2) 信息表示。指系统提供给用户信息，信息可以采用文本形式表示，也可以采用图形形式表示。

(3) 用户联机支持。指系统给用户提供的应用指导。

#### 6. 界面导航设计

界面导航所指的是如何由一个界面转换到另一个界面。可以使用活动图来描述界面之间的转换关系，其中活动图中的每一个活动状态可用来表示系统中的每一个界面。

### 习 题

1. 试述 GUI 界面设计的特点。
2. 为什么说 GUI 界面设计是一个迭代的过程？
3. GUI 界面设计中需要考虑哪些方面的用户因素？
4. 试说明 SDI 界面与 MDI 界面的区别？
5. 主要的用户交互形式有哪些？
6. 主要的用户信息表示形式有哪些？

## 第 8 章 程序算法设计与编码

程序算法设计也叫详细设计，是概要设计、界面设计完成以后需要进行的工作。程序算法设计的目标是对目标系统作出精确的设计描述，其内容包括：

- (1) 确定模块内部数据结构。
- (2) 确定模块内部程序算法。

程序算法设计结果将成为程序编码的依据。

需要注意的是程序算法设计结果不仅要求逻辑上正确，而且还要求对处理过程的设计应该尽可能简明易懂。以方便在对程序进行测试、维护时，程序具有可读性并容易理解。

### 8.1 结构化程序特征

结构化程序的基本特征是程序的任何位置是单入口、单出口的。因此，结构化程序设计中，GOTO 语句的使用受到了限制（原因是 GOTO 语句具有的随意指向特性有可能破坏结构化程序所要求的单入口和单出口特征），并且程序控制也要求采用结构化的控制结构，以确保程序的单入口和单出口特性。

结构化程序设计思想最早由 E.W.Dijkstra 提出，他在 1965 年的一次会议上指出：程序的质量与程序中所包含的 GOTO 语句的数量成反比，因此有必要从高级语言中取消 GOTO 语句。1966 年 Bohm 和 Jacopini 又证明了，只需使用图 8-1 所示的“顺序”、“选择”和“循环”这三种基本的控制结构就能实现任何复杂的程序计算问题。1972 年 IBM 公司的 Mills 进一步提出，结构化程序必须具有单入口和单出口的特征。

虽然从理论上说只使用图 8-1 中的三种基本控制结构就可以实现任何单入口单出口的程序。但是，为了方便编程和提高程序执行效率，实际程序设计中还包括一些扩展的结构化控制语句，例如：DO - CASE、DO - UNTIL、FOR - NEXT 等控制语句。

应该说，自 20 世纪 70 年代以来，结构化程序设计作为一种新的程序设计思想、方法和风格，显著地提高了软件生产率和降低了软件维护代价。1971 年 IBM 公司在纽约时报信息库管理系统的设计中成功地使用了结

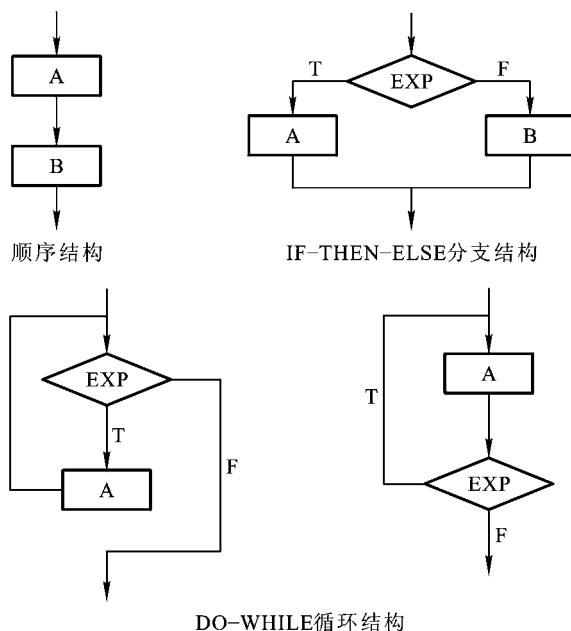


图 8-1 三种基本的结构化程序控制结构

构程序设计技术，随后在美国宇航局空间实验室飞行模拟系统的设计中，结构程序设计技术再次获得圆满成功。这两个系统都相当庞大，前者包含 8 万 3 千行高级语言源程序，后者包含 40 万行源程序，而且在设计过程中用户需求又曾有过很多改变，然而两个系统的开发工作都按时并且高质量地完成了。

结构化程序设计的基本途径是对程序问题自顶向下、逐步求精。对此，Wirth 曾做过如下说明：“我们对付复杂问题的最重要的办法是抽象，因此，对于一个复杂的程序算法问题不应该立刻用计算机指令、数字和逻辑符号来表示，而应该用较自然的抽象语句来表示，从而得出抽象程序。抽象程序对抽象的数据进行某些特定的运算并用某些合适的记号（可能是自然语言）来表示。对抽象程序做进一步的分解，并进入下一个抽象层次，这样的精细化过程一直进行下去，直到程序能被计算机接受为止。这时的程序可能是用某种高级语言或机器指令书写的。”

自顶向下、逐步求精的方法曾被应用于概要设计之中，由此可以把一个复杂的软件问题分解和细化成由许多模块组成的具有层次结构的软件系统。而在详细设计或编码阶段，通过自顶向下、逐步求精，则可以把一个模块的功能进一步分解细化为一系列具体的处理步骤，直到对应为某种高级语言的语句。

应该说，基于结构化的程序设计技术具有下述方面的优越性：

（1）自顶向下逐步求精的方法符合人类解决复杂问题的普遍规律，因此可以显著提高软件开发工程的成功率和生产率。

（2）用先全局后局部、先整体后细节、先抽象后具体的逐步求精过程开发出的程序有清晰的层次结构，因此程序容易阅读和理解。

（3）结构化的程序控制结构，能够使程序的静态结构和它的动态执行情况比较一致。程序的逻辑结构清晰，有利于程序正确性证明。即使程序出现错误也比较容易诊断和纠正。

（4）程序清晰和模块化使得在修改和重新设计一个软件时，可以更好地使程序代码得到重用。

## 8.2 程序算法设计工具

### 8.2.1 程序流程图

程序流程图又称为程序框图，其历史悠久、应用广泛，从 20 世纪 40 年代末到 70 年代中期，它一直是程序算法设计的主要工具。前面图 8-1 中对控制结构的图示，所采用的就是程序流程图。

程序流程图的主要优点是能够非常直观的描述程序的控制流程，便于初学者掌握。但是，传统的程序流程图却是一种非结构化的程序算法设计工具，并具有以下方面的一些缺陷。

（1）不能清晰地表达结构化程序所具有的控制嵌套。当控制嵌套比较复杂时，程序流程图会出现混乱。

（2）程序流程图中的箭头具有太大的随意性，例如可以通过箭头而从程序结构嵌套的最内层直接转到最外层，由此会使结构化程序所要求的单入口与单出口要求受到破坏。

（3）程序流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，

而缺少对程序全局结构的考虑。

应该说，程序流程图是一种不够健全的算法设计工具。但是，由于程序流程图历史悠久，使用人群广泛，尽管有许多人建议停止使用它，但至今仍有着应用。为了程序流程图能够适应结构化程序的要求，国际标准化组织也就专门推出了经过改进的程序流程图标记符号，如图 8-2 所示。

举例：使用程序流程图设计判断某个整数  $x$  是否为质数的算法。设计结果如图 8-3 所示。



图 8-2 程序流程图基本符号

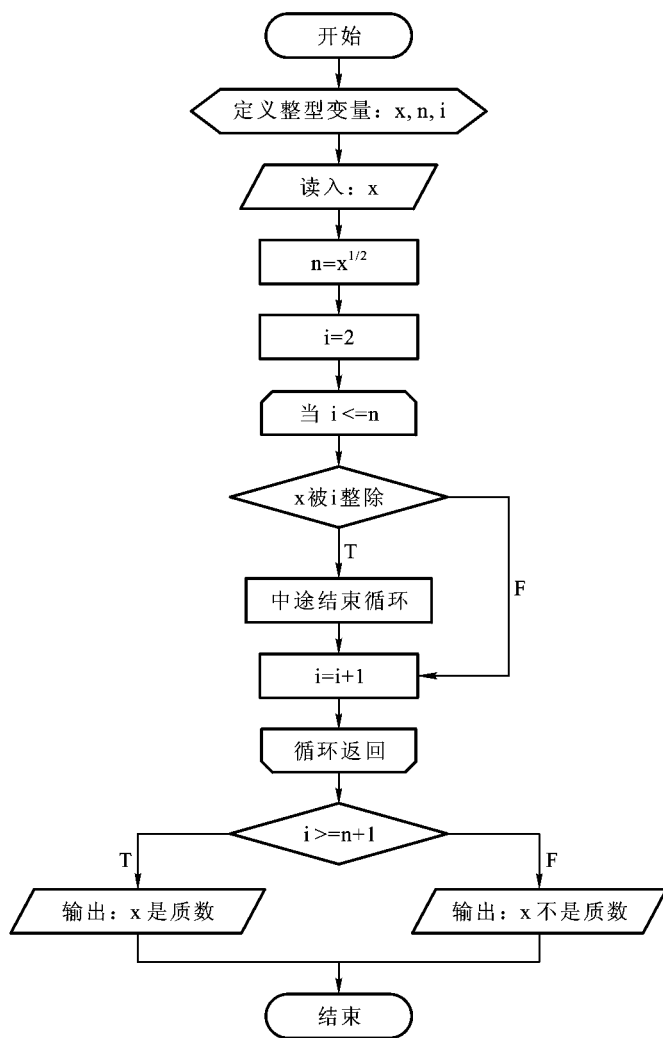


图 8-3 使用程序流程图设计程序算法

### 8.2.2 N - S 图

为了满足结构化程序设计对算法设计工具的需要，Nassi 和 Shneiderman 推出了盒图，又称为 N - S 图。它是一种严格符合结构化程序设计原则的图形描述工具。

N - S 图中用来表示控制结构的图形符号如图 8-4 所示。

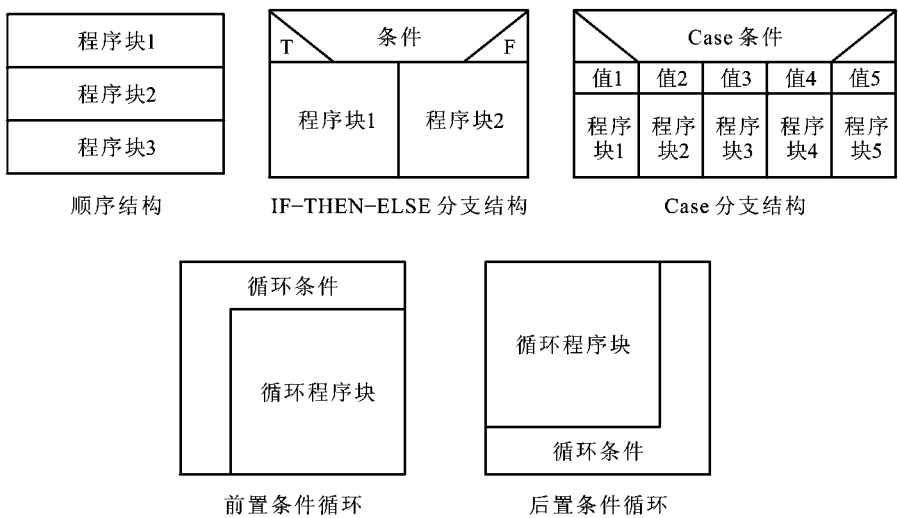


图 8-4 N - S 图基本符号

N - S 图的基本特点是通过矩形框描述模块内部程序的各个功能区域 ,并通过由外到内的矩形框嵌套表示程序的多层控制嵌套。例如图 8-5 所表示的判断某个整数 x 是否为质数的 N - S 算法设计图。

从图 8-5 可以发现 N - S 图具有以下方面的结构化特征：

- (1) 程序内部功能区域明确，每一个功能区域都被表示为一个矩形框。
- (2) 控制结构严密，不可能出现任意的转移控制。
- (3) 很容易确定局部和全程数据的作用域。
- (4) 能够清晰地表现程序内部的控制嵌套关系。

但是 ,N - S 图在获得结构严密同时,牺牲了太多的灵活性,因此不便于进行算法调整与优化。当问题比较复杂,而又采用了手工方式进行设计时,作图难度会有所增大。

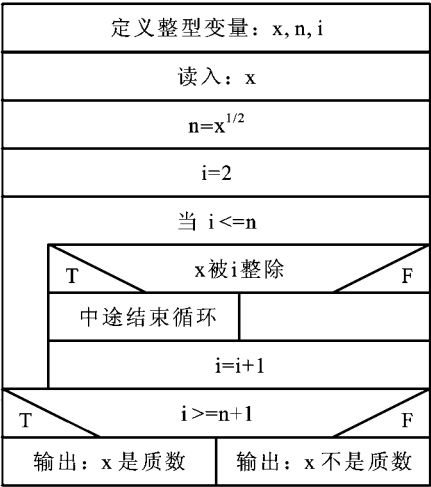


图 8-5 使用 N - S 图设计程序算法

8.2.3 PAD 图

PAD 是问题分析图 (Problem Analysis Diagram) 的英文缩写,由日本日立公司首先推出,并得到了广泛的应用。它是符合结构化程序设计原则的图形描述工具。

图 8-6 是 PAD 图的基本符号示意图。

PAD 图的基本特点是使用二维树形结构表示程序的控制流程,从上至下是程序进程方向,从左至右是程序控制嵌套关系。应该说,PAD 图与源代码有比较一致的结构,因此 PAD 图有利于由程序算法设计往高级程序语言源代码的转换。

图 8-7 是判断某个整数 x 是否为质数的 PAD 算法设计图。

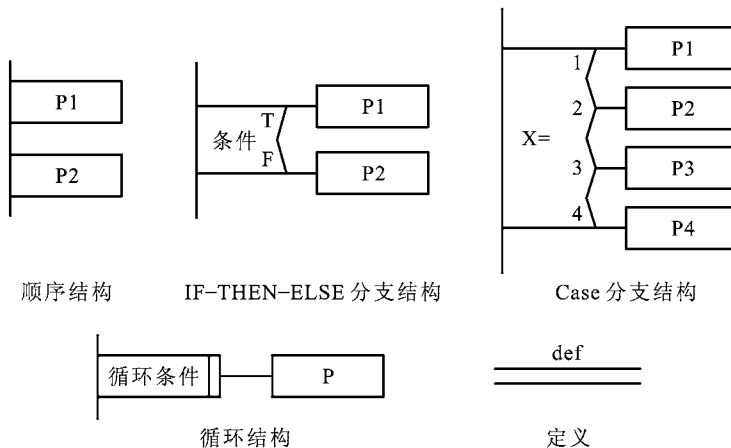


图 8-6 PAD 图基本符号

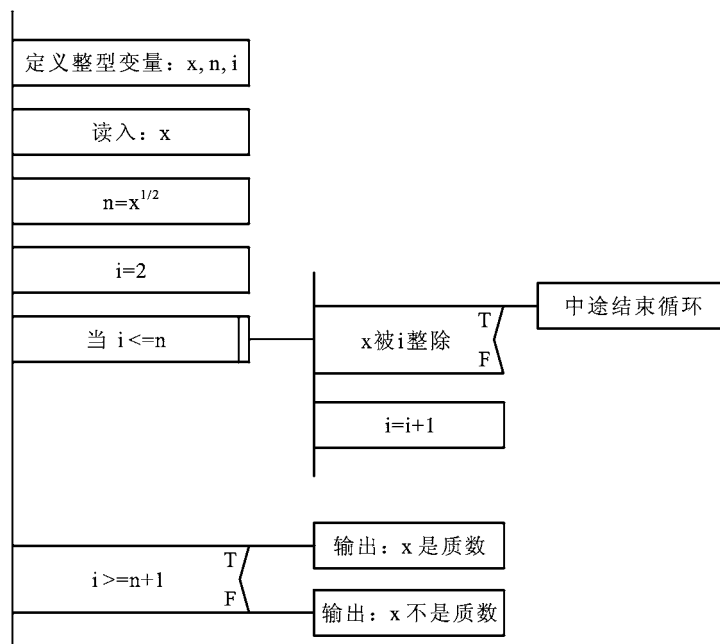


图 8-7 使用 PAD 图设计程序算法

从图 8-7 可以发现，PAD 图具有以下方面的优越性：

(1) 采用了符合结构化程序设计原则，能够满足结构化程序设计要求。

(2) 能够清晰地表现程序结构。其中，最左面的竖线是程序的主线，即第一层结构。随着程序层次的增加，PAD 图逐渐向右延伸，每增加一个层次，图形向右扩展一条竖线。PAD 图中竖线的总数就是程序的层次数。

(3) 对程序逻辑的描述具有很好的可读性，其采用二维树形结构表示程序逻辑，程序从图中最左竖线上端的结点开始执行，自上而下，从左向右顺序执行，遍历所有结点。



(4) 便于由设计向编码的过渡。可以使用软件工具自动完成由 PAD 图到高级语言源代码的转换, 由此可省去许多人工编码的工作。

(5) 支持自顶向下、逐步求精的结构化程序设计方法。开始时设计者可以仅定义一个抽象的程序框架, 随着设计工作的深入, 可以通过 def 符号逐步增加细节, 直至完成全部算法设计。

#### 8.2.4 PDL 语言

PDL 语言也称为伪码, 或过程设计语言, 它一般是某种高级语言稍加改造后的产物。由于技术背景的不同, 伪码可以是类 PASCAL、类 C 或是其他高级语言的变形。可以使用普通的正文编辑软件或文字处理系统进行 PDL 的书写和编辑。

PDL 语言的语法规则分外部语法和内部语法。其中, 外部语法用于定义程序中的控制结构和数据结构, 具有比较严谨的语法规则, 一般采用某种高级程序设计语言的语句构造, 可用于定义控制结构和数据结构; 内部语法则用于表示程序中的加工计算或条件, 它一般采用比较通俗的自然语言进行描述。因此, PDL 是一种由高级程序设计语言的框架结构和自然语言的细节描述混合而成的语言。

PDL 语言一般包括以下成分:

(1) 外部语法: 由固定的关键字提供外部语法, 它提供了结构化控制结构、数据说明和模块化的特点。为了使结构清晰和可读性好, 通常在所有可能嵌套使用的控制结构的头 and 尾都有关键字。例如, If ... EndIf、Loop ... EndLoop。

(2) 内部语法: 由自然语言提供内部语法, 用于描述程序中的加工计算。

(3) 数据说明: 需要对数据进行说明, 既包括简单数据结构 (如变量、数组), 也包括复杂的数据结构 (如结构体、链表)。

(4) 模块说明: 需要对模块功能、调用与接口等进行说明。

下面是判断某个整数  $x$  是否为质数的算法设计伪码。

PROCEDURE Verdict\_Prime

    定义整型变量:  $x, n, i$

    从键盘读入:  $x$

    给变量赋值:  $n = x^{1/2}$

    给变量赋值:  $i = 2$

DO WHILE  $i \leq n$

    IF  $x$  被  $i$  整除

    THEN

        中途结束循环

    ENDIF

$i = i + 1$

ENDDO

```

IF i >= n + 1
THEN
    输出：x 是质数
ELSE
    输出：x 不是质数
ENDIF
END Verdict_Prime

```

### 8.2.5 判定表

判定表是算法设计辅助工具，专门用于对复杂的条件组合关系及其对应的动作行为等给出更加清晰的说明，能够简洁而又无歧义地描述涉及条件判断的处理规则，并能够配合程序流程图、N - S 图、PAD 图或 PDL 伪码等进行程序算法描述。

应该说，判定表并不是一种通用的设计工具，并不能对应到某种具体的程序设计语言上去。但是，当程序算法中出现的多重嵌套中的条件选择时，往往需要用到判定表。

判定表一般由四个部分组成。其中，表的左上部分列出所有条件，表的左下部分是所有可能出现的动作，表的右上部分用于表示各种可能的条件组合，表的右下部分则是和每种条件组合相对应的动作。这样一来，判定表的右半部分的每一列实质上就构成了一条规则，它规定了与特定的条件组合相对应的动作。

下面以航空公司飞机票价计算规则为例说明判定表的使用方法。

假设该公司飞机票价计算规则是：国内乘客按规定票价计算，国外乘客按规定票价加倍计算。但中小学生凭学生证按规定票价半价优惠，70 岁以上老年乘客凭老年证或身份证按规定票价 8 折优惠，残疾人可根据残疾人证明按 8 折优惠。以上优惠可按最优折扣率计算，但不能重复计算。另外，身高不足 1 米的儿童可以免票。该飞机票价计算问题的判定表如表 8-1 所列。其中，表中右上部分中的“T”表示它左边那个条件成立，“F”表示条件不成立，空白表示这个条件成立与否并不影响对动作的选择。表的右下部分中画“Y”的标记表示做它左边的那项动作，空白表示不做这项动作。

表 8-1 用判定表描述计算飞机票价的算法

国内乘客		T	T	T	T	T	T	F	F	F	F	F	F
中小学生		F	F	T	T	F	F	F	F	T	T	F	F
老年人		F	F	F	F	T	T	F	F	F	F	T	T
残疾人		F	T	F	T	F	T	F	T	F	T	F	T
身高不足 1 米的儿童	T	F	F	F	F	F	F	F	F	F	F	F	F
免票	Y												
规定票价 * 0.5				Y	Y								
规定票价 * 0.8			Y			Y	Y						
规定票价		Y								Y	Y		
规定票价 * 1.6									Y			Y	Y
规定票价 * 2								Y					

## 8.3 Jackson 程序设计方法

1983 年法国科学家 Jackson 提出了一种以软件中数据结构为基本依据的程序算法设计方法。它是面向数据结构的，因此在以数据处理为主要内容的信息系统开发中，Jackson 程序设计方法具有一定的应用价值。

### 8.3.1 Jackson 数据结构图

Jackson 程序设计方法的基本设计途径是通过分析输入数据与输出数据的层次结构，由此对程序算法的层次结构进行推论。为了方便由数据结构映射出程序结构，Jackson 将软件系统中所遇到的数据分为顺序、选择和重复三种结构，并需要使用图形方式表示。

#### 1. 顺序结构

顺序结构的数据由一个或多个数据元素组成，每个元素按确定次序出现一次。图 8-8 是相应的数据结构图。

#### 2. 选择结构

选择结构的数据包含两个或多个数据元素，每次使用这个数据时按一定条件从这些数据元素中选择一个。图 8-9 是相应的数据结构图。

#### 3. 重复结构

重复结构的数据，根据使用时的条件由一个数据元素出现零次或多次构成。图 8-10 是相应的数据结构图。

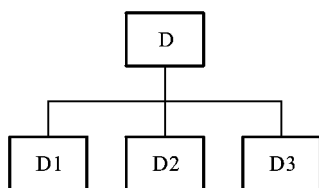


图 8-8 顺序结构

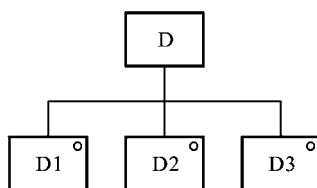


图 8-9 选择结构

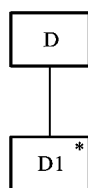


图 8-10 重复结构

### 8.3.2 Jackson 程序设计步骤

Jackson 程序结构也是顺序、选择和重复这三种结构，并可以使用与数据结构相同的图形符号表示。

Jackson 程序结构设计方法主要通过下述五个步骤进行，由此可实现由数据结构到程序结构的映射。

1. 分析并确定输入数据和输出数据的逻辑结构，并用 Jackson 数据结构图描绘这些数据。

2. 找出输入数据结构和输出数据结构中有直接对应关系的数据单元。这其中的数据直接对应关系是指输出数据与输入数据有直接的因果关系，在程序中可以同时处理的数据单元（对于重复出现的数据单元必须重复的次序和次数都相同才可能有对应关系）。

3. 用下述三条规则从描绘数据结构的 Jackson 图中导出描绘程序结构的 Jackson 图：

第一，为每对有直接对应关系的数据单元，按照它们在数据结构图中的层次在程序结构图的相应层次上画一个处理框。其中，如果这对数据单元在输入数据结构和输出数据结构中所处的层次不同，则和它们对应的处理框在程序结构图中所处的层次与它们之中在数据结构图中层次低的那个对应。

第二，根据输入数据结构中剩余的每个数据单元所处的层次，在程序结构图的相应层次分别为它们画上对应的处理框；

第三，根据输出数据结构中剩余的每个数据单元所处的层次，在程序结构图的相应层次分别为它们画上对应的处理框。

4. 列出所有操作和条件，包括分支条件和循环结束条件，并且把它们分配到程序结构图的适当位置上去。

5. 用伪码表示程序。其使用的伪码和 Jackson 图是完全对应的，下面是和三种基本结构对应的伪码。

#### (1) 顺序结构

```
A seq  
  B  
  C  
  D  
A end
```

其中的 'seq' 和 'end' 是关键字。

#### (2) 选择结构

```
A select cond1  
  B  
A or cond2  
  C  
A or cond3  
  D  
A end
```

其中的 'select'、'or' 和 'end' 是关键字，cond1、cond2 和 cond3 是执行 B、C 或 D 时需要满足的条件。

#### (3) 重复结构

```
A iter until (或 while) cond  
  B  
A end
```

其中的 "iter"、"until"、"while" 和 "end" 是关键字，cond 是条件。

### 8.3.3 Jackson 程序设计举例

假设某销售管理系统需要根据销售员注册数据表和销售员销售情况记录数据表，按销售员代号产生销售汇总报表。需要输入的两个数据表中，销售员注册数据表为主表，销售员销售情

况记录数据表为从表，这两个数据表结构如图 8-11 所示。

销售员注册表: 表		
	销售员编号	销售员姓名
▶ +	1	黎斌
+	2	王志明
+	3	张兰
*	(自动编号)	
记录: 1		

销售员销售情况记录表: 表			
	销售序号	销售员编号	销售额
▶	1	1	¥ 1,200.00
	2	1	¥ 2,300.00
	3	2	¥ 2,100.00
	4	1	¥ 6,700.00
	5	3	¥ 4,000.00
	6	2	¥ 3,860.00
	7	3	¥ 1,680.00
	8	2	¥ 5,600.00
	9	1	¥ 8,690.00
	10	2	¥ 6,690.00
*	(自动编号)	0	¥ 0.00
记录: 1 共有记录数: 10			

图 8-11 需要输入的两个数据表

需要产生的销售数据汇总报表格式如下：

销售数据汇总表		
编号	姓名	累计销售额
1	黎斌	¥ 18,890.00
2	王志明	¥ 18,250.00
3	张兰	¥ 5,680.00
销售额：¥ 42,820.00		

Jackson 程序设计方法的第一步是确定输入输出数据结构。这是一个从数据表输入数据，然后通过报表输出数据的问题。其中 ,需要输入的数据表分为主表和从表。图 8-12 是用 Jackson 图描绘的输入输出数据结构。

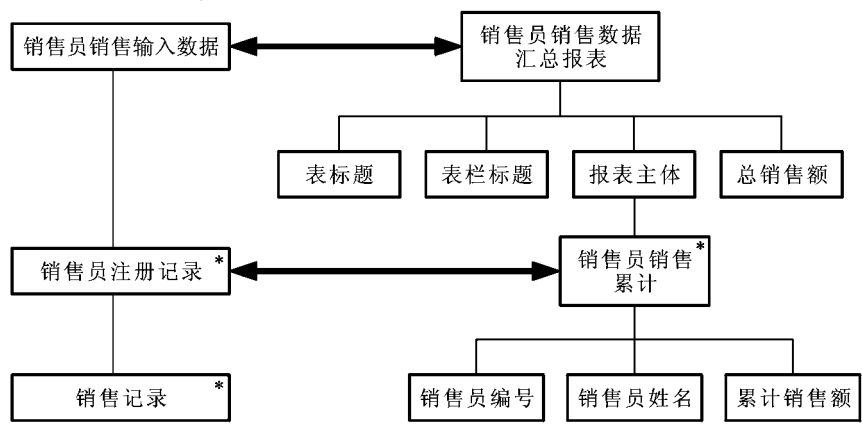


图 8-12 输入输出数据结构及其因果对应关系

在输入输出数据结构确定之后，接着需要考虑的是输入输出数据之间存在的因果对应关系。由于输出数据总是通过对输入数据的处理而得到的，因此在输入输出数据结构最高层次的

两个单元，即“销售员销售输入数据”与“销售员销售数据汇总报表”之间总是存在因果对应关系。另外，由于销售汇总时的累计单位是销售员，因此“销售员注册记录”与“销售员销售累计”之间也存在着因果关系。这些数据之间存在的因果对应关系也在图 8-12 中使用双向箭头表示出来了。

Jackson 程序设计方法的第三步是从数据结构图导出程序结构图。按照 8.3.2 节介绍的规则，这个步骤的大致过程是：

(1) 为每对有直接对应关系的输入输出数据单元，按照它们在数据结构图中的层次，在 Jackson 程序结构图的相应层次上画一个处理框，由此确定 Jackson 程序结构图的关键节点。如程序结构图的最顶层的处理框“销售员销售数据汇总报表打印程序”，它与“销售员销售输入数据”和“销售员销售数据汇总报表”这对最顶层的数据单元相对应；“以销售员注册记录为单位产生销售累计”，它与“销售员注册记录”和“销售员销售累计”这两个数据单元相对应。

(2) 根据输入输出数据结构中剩余的每个数据单元所处的层次，在程序结构图的相应层次分别为它们画上对应的处理框。如“打印表标题”、“打印表栏标题”、“产生表体”、“打印总销售额”、“打印销售员编号”、“打印销售员姓名”、“累计销售额”和“处理销售记录”。

根据上述两步可以得到图 8-13 所示的 Jackson 程序结构基本框架图。

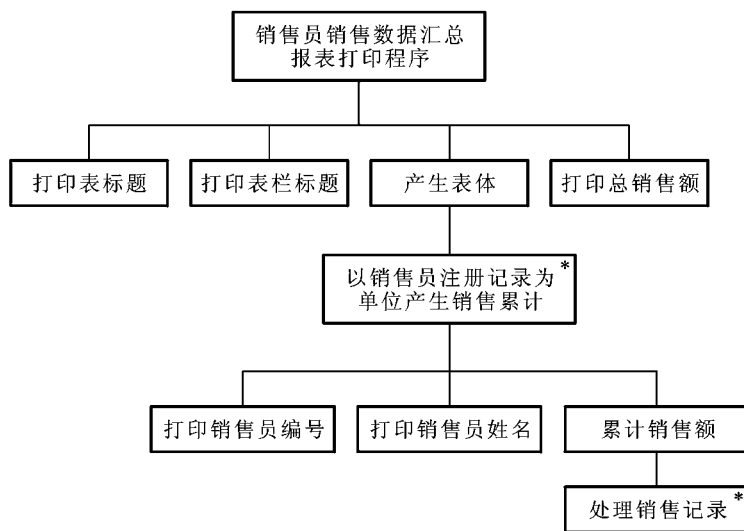


图 8-13 Jackson 程序结构基本框架

(3) 图 8-13 所反映的还只是 Jackson 程序结构的基本框架，接下来还需要将程序运行中需要涉及的其他操作步骤添加到程序结构图里去。例如，“打开销售员注册表”、“关闭销售员注册表”、“按销售员编号读入销售记录集”、“关闭销售记录集”、“将累计销售额累加到总销售额”、“移到下一条销售员记录”、“将当前记录的销售额累加到累积销售额”、“移到下一条销售记录”、“打印累计销售额”、“将累计销售额累加到总销售额”和“移到下一条销售员注册记录”等。由此可以得到比较完整的 Jackson 程序结构图，如图 8-14 所示。

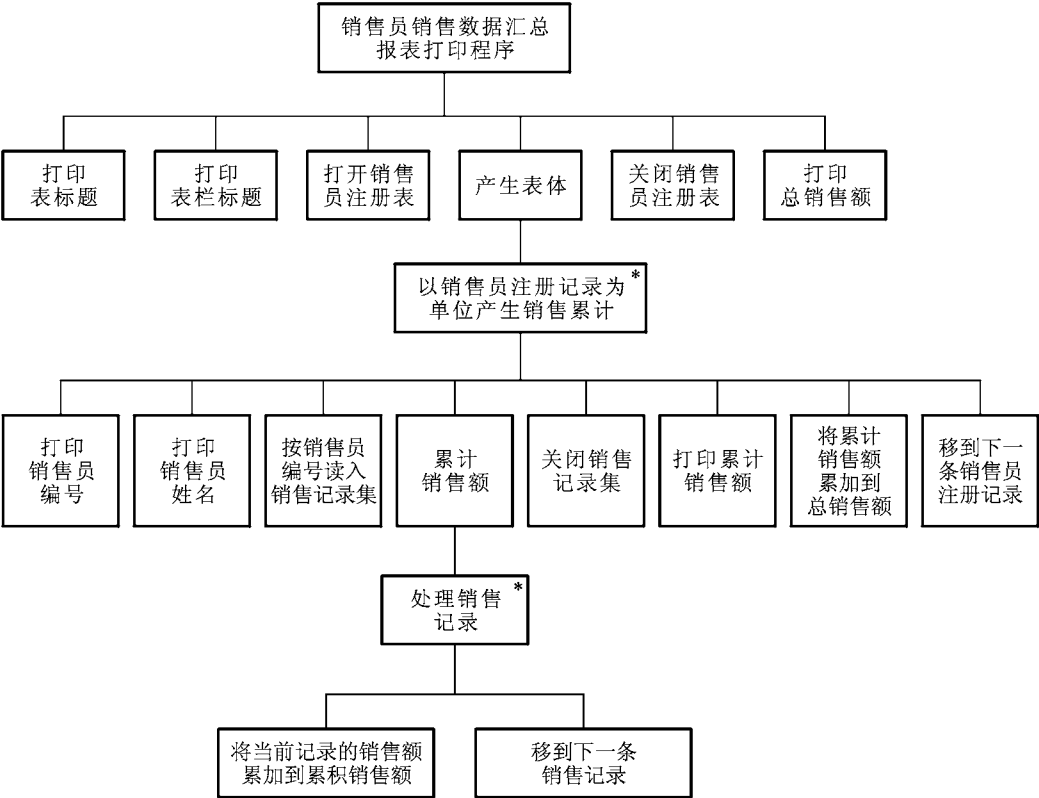


图 8-14 将其他操作步骤添加进去之后的 Jackson 程序结构图

Jackson 程序设计方法的最后一步是用伪码表示程序处理过程。因为 Jackson 使用的伪码和 Jackson 图之间存在简单的对应关系，所以根据图 8-14，可以很便利地得出下面的伪码程序过程：

```
销售员销售数据汇总表打印程序 seq
    打印汇总表标题
    打印汇总表栏标题
    打开销售员注册表
    产生表体 iter until 销售员注册表底
        以销售员注册记录为单位产生销售累计 seq
            打印销售员编号
            打印销售员姓名
            按销售员编号读入销售记录集
            累计销售额 iter until 销售记录集底
                处理销售记录 seq
                    将当前记录的销售额累加到累积销售额
                    移到下一条销售记录
                处理销售记录 end
            累计销售额 end
```

```
        关闭销售记录集
        打印累计销售额
        将累计销售额累加到总销售额
        移到下一条销售员注册记录
        以销售员注册记录为单位产生销售累计 end
    产生表体 end
    关闭销售员注册表
    打印总销售额
销售员销售数据汇总表打印程序 end
```

## 8.4 程 序 编 码

编码是软件工程过程中需要经历的一个步骤。在完成程序算法设计之后，接着需要编码。直到今天，计算机的工作仍然依赖于通过某种程序设计语言进行，因此，程序编码也就自然是软件工程过程中必须经历的一个步骤。在完成程序算法设计之后，接着就需要进行编码。

程序编码的目的是实现人和计算机的通信，指挥计算机按照人的意志正确工作。尽管软件的质量更多地依赖于软件设计，但是，程序设计语言的特性和编码途径等，也会对程序的可靠性、可读性、可测试性和可维护性产生影响。

### 8.4.1 编程语言种类

#### 1. 低级语言

低级语言包括第一代机器语言与汇编语言，它们是直接面向机器的语言。

应该说，自从有了计算机，就有了机器语言，并且是惟一能够直接在机器上运行的语言，使用汇编语言、高级语言编写的程序，最终都要经过汇编或编译转变为机器代码之后，才能在计算机上运行。

机器语言由机器指令组成，并且机器不同则其机器语言指令也就不同。用机器语言编写的程序，都是二进制代码的形式，所有的地址分配都是以绝对地址的形式处理。假如使用机器语言编码，则存储空间的安排，寄存器、变址的使用等，都需要由程序员自己计划，因此使用机器语言编写程序容易出错，目前已很少直接使用机器语言编码。

汇编语言是符号指令方式代替机器语言中的二进制指令，因此它比机器语言直观。汇编语言的每一条符号指令都与相应的机器指令有对应关系，同时又增加了一些诸如宏、符号地址等功能，其存储空间的安排也可由机器解决，因此相应减少了程序员的工作量，也减少了出错率。

但汇编语言也是面向机器的，可以说不同指令集的处理系统就有自己相应的汇编语言，基本上有多少种型号的计算机也就有多少种汇编语言。为使软件具有较好的可移植性，从软件工程的角度来看，只在高级语言无法满足设计要求的时候，例如不具备支持某种特定的功能，或达不到所要求的技术性能的时候，汇编语言才被使用。

#### 2. 高级语言

高级语言是指面向问题求解过程的语言，它们使用了与人的思维体系更加接近的概念和符



号。高级语言中的一条语句往往对应着若干条机器指令。通常情况下,高级语言不依赖于实现这种语言的计算机,因此具有较好的可移植性。

高级语言最早出现在 20 世纪 50 年代末期,如 FORTRAN、COBOL。1958 年诞生了最早的高级语言是 FORTRAN,1959 年又出现了 COBOL,20 世纪 60 年代中期有了 BASIC,20 世纪 70 年代初期有了 C 语言与 PASCAL 语言,20 世纪 80 年代初期产生了 C++ 语言,20 世纪 90 年代初期产生了 Java 语言。

为了更好地认识高级语言,有必要对它进行分类。从高级语言的发展历史来看,可以将它分为传统高级语言、结构化高级语言、4GL 高级语言和面向对象高级语言。

### (1) 传统高级语言

传统高级语言是指一些早期高级语言,例如 FORTRAN、COBOL 和 BASIC。它们历史悠久、应用广泛,已被广泛人群了解和接受。

FORTRAN 语言是出现最早的高级语言,诞生于 1958 年。早期 FORTRAN 语言主要用于解决科学与工程计算,具有较强的计算能力,但它缺乏结构化所需的一些基本控制结构,并且数据类型贫乏,因此不太适应商业方面的数据处理。

COBOL 语言也是早期出现的高级语言,诞生于 1959 年。在一个比较长的时期里,COBOL 语言是商用方面数据处理应用的广泛使用的标准语言。它具有极强的数据定义能力,数据描述与算法描述分开,结构严谨、层次分明,并且词汇、句型、语法等与英语接近,因此可读性强。由于 COBOL 语言与硬件环境分离,这使得它具有较好的可移植性。

BASIC 语言产生于 20 世纪 60 年代中期,它原是一种解释型语言,是为在分时方式下讲授程序设计而研制。在 20 世纪 70 年代,经过改版的 Turbo BASIC 语言具有了像 PASCAL 语言、C 语言一样的结构化语言特征,并在 PC 机上得到广泛应用。在 20 世纪 80 年代,又产生了具有面向对象与可视开发特征的 Visual Basic,它具有图形设计工具、结构化的事件驱动编程模式、开放的环境,使用户可以既快又简便地编制出 Windows 环境下的各种应用程序。

### (2) 结构化高级语言

结构化高级语言产生于 20 世纪 70 年代初期,它们具有结构化控制结构,并具有较强的过程构造能力和数据结构能力,例如 PASCAL 语言、C 语言。由于结构化语言所具有的编程与维护方面的优势,因此许多传统的高级语言,例如 FORTRAN、COBOL、BASIC 等,都曾转变成为结构化高级语言。

Pascal 是 70 年代初期研制出来的结构化程序设计语言。主要用于结构化程序设计的教学,具有一系列结构化程序设计语言的特点,例如程序块结构、强数据类型、直接递归功能等。各种计算机系统都有配套的 Pascal 语言,IBM PC 机上比较受欢迎的是美国 Borland 公司的 Turbo Pascal 语言,它提供了一种全新的集成开发环境,支持鼠标、多文件编辑和多重叠窗口,并配备了增强型调试工具和功能完备的嵌入式汇编器,可用来编写微机上任何类型的程序,调用其他高级语言程序和被其他高级语言程序调用的能力都很强。

C 语言最初是作为设计操作系统的语言而研制的,UNIX 操作系统就是用 C 语言实现的。C 语言具有很强的功能而且十分灵活,它支持复杂的数据结构,可大量应用指针,并有丰富的运算操作符和数据处理操作符。此外,它还具有类似汇编语言的特性,使程序员能够直接操作机器地址,C 语言也因此具有了“中级语言”的美誉。IBM PC 机上比较流行的 C 版本是美国 Borland

公司的 Turbo C，它是一个高效、优化的 C 编译程序，支持 ANSI C 标准，支持集成环境和命令行调试。在编程过程中，Turbo C 具有单步执行、设置断点、监视和计算表达式等功能，并提供了一个丰富的图形函数库。

### (3) 面向对象高级语言

最早的面向对象高级语言是 Smalltalk，它引入了与传统程序设计语言根本不同的控制结构与数据结构，可以通过类创建对象实例，并可以通过类的继承性而使新对象方便地获得类的属性，由此使程序代码具有较好的重用性。

在 Smalltalk 之后，C++ 成为最受欢迎的面向对象的程序设计语言。包括 Turbo C++、Borland C++、Microsoft Visual C++ 等。应该说，C++ 既融合有面向对象的能力，又与 C 语言兼容，保留了 C 语言的许多重要特性。这就维护了大量已开发的 C 库、C 工具以及 C 源程序的完整性，使 C 程序员不必放弃自己已经十分熟悉的 C 语言，而只需要补充学习 C++ 提供的那些面向对象的概念即可。

Java 语言产生于 20 世纪 90 年代初期，由 SUN 公司最先推出，它是一种基于 Java 虚拟机的解释型语言，因此具有跨操作平台的能力，能够工作在 UNIX、Linux、Windows 等各种不同的操作系统上。Java 具有面向对象、分布式应用、中间编译解释、与操作系统无关、便于移植、多线程、动态处理等诸多特点，Java 基本结构接近 C++ 语言，但做了许多重大修改。它不再支持运算符重载、多继承及许多自动强制等易混淆和较少使用的特性，增加了内存空间自动垃圾收集的功能，因此，比较起 C++ 来，Java 更加安全、可靠。Java 还提供了附加的例程库，通过它们的支持，Java 应用程序能够自由地打开和访问网络上的对象，就像在本地文件系统中一样。因此，Java 程序特别适合于 Internet 环境，其对网络应用的支持，有力地推动了 Internet 和企业网络的 Web 应用的发展。

### 3. 第四代语言 (4GL)

第四代语言 (4GL) 是指一些面向问题的高级语言，例如 SQL 结构化数据查询语言。同其他人工语言一样，第四代语言也用不同的文法表示程序结构和数据结构，但是第四代语言是在更高级抽象的层次上表示这些结构，它不再需要规定程序算法细节。

应该说，4GL 语言兼有过程性和非过程性的两重特性。程序员规定条件和相应的动作，这是过程性的部分，并且指出想要的结果，这是非过程部分。然后由 4GL 语言系统运用它的专门领域的知识来填充过程细节。

Martin 把 4GL 语言分为以下几种类型：

(1) 查询语言：通常查询语言是为与数据库有关的应用而开发的。用户可以利用查询语言对预先定义在数据库中的信息进行较复杂的操作。

(2) 程序生成器：程序生成器只需很少的语句就能生成完整的第三代语言程序，它不必依赖预先定义的数据库作为它的着手点。

(3) 其他 4GL：除了查询语言和程序生成器以外，还有其他一些类型的 4GL 语言。例如判定支持语言、原型语言、形式化规格说明语言等。

## 8.4.2 选择编程语言的依据

在对软件系统进行编码之前，必须做出一个重要抉择，这就是使用什么样的程序设计语言

实现这个软件系统。由于适宜的程序设计语言能使根据设计去完成编码时困难最少，并可以减少程序测试，还可以使源程序更容易阅读和更容易维护，因此需要认真对待。

通常情况下，高级语言比汇编语言具有更大的优势。因此，除了在很特殊的应用领域，或者系统中执行时间非常关键的，或直接依赖于硬件设备的一小部分代码，它们需要用汇编语言书写之外，软件中的其他程序大都应该使用高级语言书写。

那么，在种类繁多的高级语言中选择哪一种呢？对此，需要从软件问题技术角度、工程角度以及程序员心理学角度等多个方面对程序设计语言进行评价，比较各种语言的适用程度，考虑语言的现实可能性等。例如，对于一种新型语言工具的考虑。显然，新的语言工具有更强大的功能，对于应用有很强的吸引力，但是因为已有的语言已经积累了大量的久经使用的程序，具有完整的资料、支撑软件和软件开发工具，程序设计人员已比较熟悉，而且有过类似项目的开发经验和成功的先例，由于心理因素，开发人员往往会更愿意选用原有的语种。所以对是否选择这种新型语言工具，应当进行更全面地分析与评价。

实际上，在选择编程语言时往往需要考虑诸多方面的因素，例如以下方面的因素：

- (1) 软件项目的应用领域。
- (2) 软件问题的算法复杂性。
- (3) 软件的工作环境。
- (4) 软件在性能上的需要。
- (5) 软件中数据结构的复杂性。
- (6) 软件开发人员的知识水平和心理因素等。

在以上因素中，项目的应用领域是需要着重考虑的最关键因素。

高级编程语言有偏高与偏低的区别，例如，Visual Basic 是偏高的程序设计语言，而 Visual C++ 则是偏低的程序设计语言。通常情况下，偏向高层或客户端应用的软件项目、投入资金过少的软件项目、需要在短期内投入使用的软件项目适合选用偏高的程序设计语言，而偏向低层或服务器端应用的软件项目、与硬件设备有关的项目、注重于研究的软件项目等则适合选用偏低的程序设计语言。

有时候，一个软件项目中还可能需要用到多种编程语言，例如，可以将 Visual Basic 与 Visual C++ 结合起来应用，其中 Visual Basic 用于前端界面开发，而 Visual C++ 则用于后台服务器组件开发。

### 8.4.3 编程风格与质量

编程风格是编写程序时需要遵守的一些规则。在衡量程序质量时，源程序代码的逻辑简明清晰、易读易懂是一个重要因素，而这些都与编程风格有着直接的关系，例如程序格式、程序中的注释、数据说明方式、输入输出方式等。

#### 1. 源程序文档化

所谓源程序文档化是指在程序中恰当地使用标识符、注解，并注意程序的视觉组织等，由此使源程序便于阅读、容易理解。对此，需要注意以下几点规则：

(1) 注意标识符的命名。选取含义鲜明的名字，使它能正确地提示程序对象所代表的实体，这对于帮助阅读者理解程序是很重要的。如果使用缩写，那么缩写规则应该一致，并且应该给

每个名字加注解。

(2) 注意注解对程序的有意义的解释。正确的注解非常有助于对程序的理解。通常在每个模块开始处有一段序言性的注解, 简要描述模块的功能、主要算法、接口特点、重要数据以及开发简史。插在程序中间与一段程序代码有关的注解, 主要解释包含这段代码的必要性。对于用高级语言书写的源程序, 不需要用注解的形式把每个语句翻译成自然语言, 应该利用注解提供一些额外的信息。应该用空格或空行清楚地区分注解和程序。注解的内容一定要正确, 错误的注解不仅对理解程序毫无帮助, 反而会妨碍对程序的理解。

(3) 注意程序清单的布局。程序清单的布局是一种非常重要的视觉组织, 对于程序的可读性也有很大影响, 应该利用适当的阶梯形式使程序的层次结构清晰明显。

例如, 下面这段使用 Java 语言编写的程序。

```
//-----  
// 模块名: DateBean          完成日期: 2003-8-26  
// 开发机构: ABC 软件公司    设计人: 黎斌  
// -----  
  
// 引用类库  
import java.awt.*;  
import java.util.Calendar;  
import javax.swing.*;  
  
public class DateBean extends JLabel{  
    // 定义变量  
    private Calendar c = Calendar.getInstance();  
    private int m, d, y, e;  
    private String month, day, year, era, month_str;  
    private boolean useMonthString;  
    private Color fontColor;  
    private int style;  
  
    // 定义常量  
    public static final int MONTH_DAY_YEAR = 1;  
    public static final int YEAR_MONTH_DAY = 2;  
    public static final String[] allMonths = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",  
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};  
  
    public DateBean() {  
        m = c.get(Calendar.MONTH) + 1;  
        d = c.get(Calendar.DATE);
```

```
y = c.get(Calendar.YEAR);
e = c.get(Calendar.ERA);

if(m <= 9)
    month = "0" + String.valueOf(m);
else
    month = String.valueOf(m);

if(d <=9)
    day = "0" + String.valueOf(d);
else
    day = String.valueOf(d);

year = String.valueOf(y);
month_str = allMonths[m - 1];
}

// 设置字体颜色与字体型号
public void setFontColor(Color newFontColor) {
    super.setForeground(newFontColor);
}

public Color getFontColor() {
    return super.setForeground();
}

public void setFont(Font newFont) {
    super.setFont(newFont);
}

public Font getFont() {
    return super.getFont();
}

// 设置背景颜色
public void setBackground(Color newBackground) {
    super.setBackground(newBackground);
}
```

```
public Color getBackground() {
    return super.getBackground();
}

// 设置日期格式
public void setStyle(int newStyle) {
    style = newStyle;

    switch(style){
        case MONTH_DAY_YEAR:
            if(useMonthString)
                this.setText(month_str + " " + day + ", " + year);
            else
                this.setText(month + " / " + day + " / " + year);
            break;
        case YEAR_MONTH_DAY:
            if(useMonthString)
                this.setText(year + " " + month_str + " " + day);
            else
                this.setText(year + " / " + month + " / " + day);
            break;
        default:
            this.setText("invalid");
    }
}

public int getStyle() {
    return style;
}

public void setUseMonthString( boolean x ){
    useMonthString = x;
    this.setStyle(this.getStyle());
}

public boolean getUseMonthString(){
    return useMonthString;
}
}
```

## 2. 数据说明

虽然在设计期间已经确定了数据结构的组织和复杂程度,然而数据说明的风格却是在写程序时确定的。为了使数据更容易理解和维护,有一些比较简单的原则应该遵循。

数据说明的次序应该标准化,例如,按照数据结构或数据类型等,确定说明的次序。有次序就容易查阅,因此能够加速测试、调试和维护的过程。

当多个变量名在一个语句中说明时,应该按字母顺序排列这些变量。

如果设计时使用了一个复杂的数据结构,则应该用注解说明使用程序设计语言实现这个数据结构的方法和特点。

## 3. 语句构造

虽然软件设计期间已经确定了软件的逻辑结构,然而程序中的语句的构造却只能在编写程序时确定。构造语句时应该遵循的原则是,每个语句都应该简单而直接,不能为了提高效率而使程序变得过分复杂。

下述规则有助于使语句简单明了:

- (1) 不要为了节省空间而把多个语句写在同一行。
- (2) 尽量避免使用复杂的条件组合表达式。
- (3) 尽量避免在条件表达式中使用逻辑“非”运算。
- (4) 尽量避免大量使用循环嵌套和条件嵌套。
- (5) 利用括号使逻辑表达式或算术表达式的运算次序清晰直观。

## 4. 输入与输出

在设计和编写程序时应该考虑下述有关输入与输出方面的规则:

- (1) 对所有输入数据都需要进行检验。
- (2) 检查输入项重要组合的合法性。
- (3) 尽量使用操作简单的输入格式。
- (4) 在遇到多项数据输入时,应该使用数据结束标记。
- (5) 明确提示交互式输入的请求,并详细说明可用的选择或边界数值。
- (6) 当程序设计语言对格式有严格要求时,应保持输入格式的一致性。

### 8.4.4 影响程序工作效率的因素

一般说来,程序工作效率会受到处理器计算速度、存储器存储容量和输入输出速度等几个方面因素的影响,并与程序设计语言、操作系统、硬件环境等有着直接关系。因此,在考虑程序工作效率时,需要将诸多因素综合起来分析。

#### 1. 评估程序效率时需要遵循的准则

应该说,程序的工作效率是衡量程序质量时的一个重要指标,并特别受到一些高要求用户的重视。尽管程序工作效率非常重要,但在对其做评估,或考虑提高程序工作效率时,则又不得不进行利弊权衡,对此,以下几点值得注意:

(1) 程序工作效率是系统性能要求,因此应该是在需求分析阶段确定下来,软件开发需要做到的是达到需求所要求的效率,而不应该是去追求最高效率。

(2) 程序工作效率主要是靠好的设计来提高的,编码对程序工作效率的改善,相对而言比

较有限。

(3) 程序的效率和程序的简单程度是一致的，不要为了提高程序工作效率而牺牲程序的清晰性和可读性，这将可能导致因小而失大。

#### 2. 对程序运算速度的考虑

源程序的效率直接由详细设计阶段确定的算法的效率决定，但是，写程序的风格也能对程序的执行速度和存储器要求产生一定的影响。对此需要注意下述规则：

(1) 尽量简化算术的和逻辑的表达式。

(2) 仔细研究嵌套的循环，以确定是否有语句可以从内层移到外层。

(3) 尽量避免使用多维数组。

(4) 尽量避免使用指针和复杂的表。

(5) 使用执行时间短的算术运算，如整数运算。

(6) 不要混合使用不同的数据类型。

(7) 尽量使用整数运算和布尔表达式。

(8) 在效率是决定性因素的应用领域，应该尽量使用具有良好优化特性的编译程序，以求能生成高效的目标代码。

#### 3. 对存储器效率的考虑

在大型计算机中，必须考虑操作系统页式调度的特点，一般说来，使用能保持功能域的结构化控制结构，是提高效率的好方法。

在微处理机中，如果要求使用最少的存储单元，则应选用有紧缩存储器特性的编译程序，在非常必要时可以使用汇编语言。

提高执行效率的技术通常也能提高存储器效率。提高存储器效率的关键同样是“简单”。

#### 4. 对输入、输出效率的考虑

如果用户为了给计算机提供输入信息或为了理解计算机输出的信息，所需花费的脑力劳动是经济的，那么人和计算机之间通信的效率就高。因此，简单清晰同样是提高人机通信效率的关键。

硬件之间的通信效率是很复杂的问题，但是，从编写程序的角度来看，却有些简单的原则可以用来提高输入与输出效率。例如以下规则：

(1) 所有输入与输出都应该有缓冲，以减少用于通信的额外开销。

(2) 对二级存储器（如磁盘）应选用最简单的访问方法。

(3) 二级存储器的输入与输出应该以信息组为单位进行。

(4) 如果“超高效的”输入与输出很难被人理解，则不应采用这种方法。

## 8.5 程序算法复杂性度量

程序算法复杂性主要指模块内程序的复杂性，它直接关系到软件开发费用的多少，开发周期的长短和软件内部潜伏错误的多少等。同时它也是软件可理解性的另一种度量。而通过定量度量程序的复杂程度，可以估算出软件中故障的数量以及软件开发需要用的工作量，并且还可以通过定量度量结果，比较两个不同的设计或两个不同算法的优劣。

比较著名的程序算法复杂性度量方法是 McCabe 度量法，其由 Thomas McCabe 提出，这是



一种基于程序控制流的复杂性度量方法。

McCabe 方法是通过计算程序中环路的个数来估算程序复杂程度的，因此，为了使用 McCabe 方法来进行程序算法复杂性度量，则首先需要画出程序图。

所谓程序图，也就是退化了的传统程序流程图，其仅仅表现程序内部的控制流程，而并不涉及对数据的具体操作以及分支或循环的具体条件。

通过把传统程序流程图中每个处理符号都退化成一个结点，并把原来连接不同处理符号的箭头变成连接不同结点的有向弧，这样就可以得到程序图。例如图 8-16 中的程序图，就是从图 8-15 中的程序流程图转换而来的。

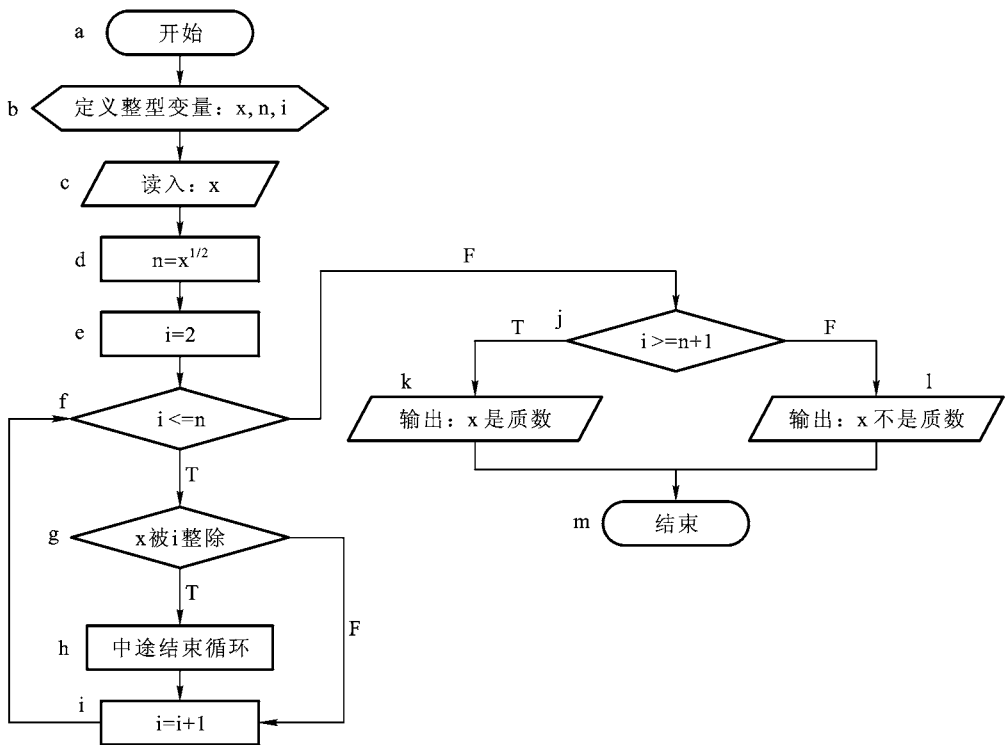


图 8-15 传统程序流程图

McCabe 方法对程序复杂性的度量，所采用的是程序的环形复杂度，它等于强连通的程序图中线性无关的有向环的个数。

根据图论，在一个强连通的有向图 G 中，环的个数由以下公式得出；

$$V(G) = m - n + p$$

其中， $V(G)$  是有向图 G 中的环数， $m$  是有向图 G 中的弧数， $n$  是有向图 G 中的节点数， $p$  是有向图 G 中分离部分的数目。

对于一个正常的程序，总应该能够从程序图的起点到达图中任何一个节点，假如某个节点不能到达，则意味着这个节点上的程序代码永远不能被执行，显然这是一个存在错误的不正常的程序。因此，正常程序的程序图总是连通的，也就是说它的  $p = 1$ 。

所谓强连通图则是指从程序图中任一个节点出发，可以到达其需要到达的任何其他节点。

程序图通常不是强连通的，因为从图中较低的节点（即较靠近结束点的节点）往往不能到达较高的节点。但是如果从结束点到开始点补画一条虚弧，则程序图就必然会成为强连通的。

得出上述结论有以下三点理由：

- (1) 从开始点总能到达图中任何一点；
- (2) 从图中任何一点总能到达结束点；
- (3) 经过从结束点到开始点补画的弧，可以从结束点到达开始点。

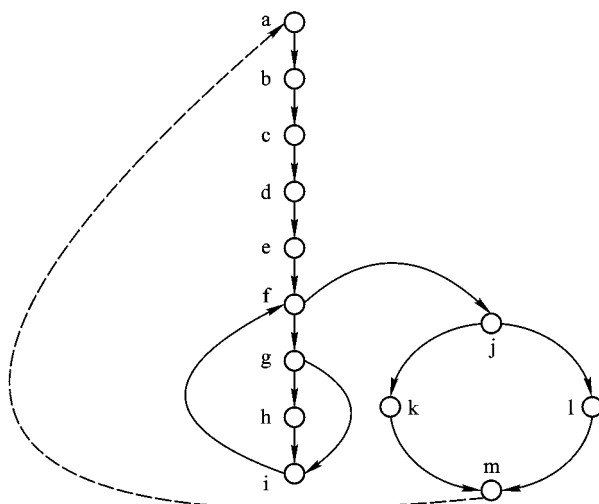


图 8-16 程序图

图 8-16 就从结束节点 m 到开始节点 a 之间，补画了一条虚弧线，这样它就成为了强连通的图。而在图 8-16 这个强连通图中，节点数为 13，弧数为 16，因此其环形复杂度是：

$$V(G) = 16 - 13 + 1 = 4$$

计算环形复杂度还有其它方法。对于平面图而言，环形复杂度等于强连通的程序图在平面上围成的区域的个数。而图 8-16 中的强连通过程图有 4 个被围成的封闭区域，因此其复杂度  $V(G) = 4$ 。

程序的环形复杂度取决于程序控制流的复杂程度。当程序内分支数或循环个数增加时，环形复杂度也随之增加，因此它是对测试难度的一种定量度量，也能对软件最终的可靠性给出某种预测。

McCabe 研究大量程序后发现，程序环形复杂度越高，则程序也就越复杂，往往越难实现，越容易出问题。对此，McCabe 建议，模块内程序复杂度应该限制在  $V(G) < 10$  以内。对于复杂度超过 10 的程序，则应该将程序分成几个小程序，由此以减少程序复杂度，并可以使程序出错的可能性降低。

## 小 结

### 1. 结构化程序特征

结构化程序的基本特征是程序的任何位置是单入口、单出口的。因此，结构化程序设计中，

GOTO 语句的使用受到了限制,并且程序控制也要求采用结构化的控制结构,以确保程序是单入口和单出口的。

## 2. 程序算法设计工具

### (1) 程序流程图

程序流程图又称为程序框图,其历史悠久、应用广泛,从 20 世纪 40 年代末到 70 年代中期,它一直是程序算法设计的主要工具。程序流程图的主要优点是能够非常直观的描述程序的控制流程。但是,传统的程序流程图却是一种非结构化的程序算法设计工具。

### (2) N - S 图

为了满足结构化程序设计对算法设计工具的需要,Nassi 和 Shneiderman 推出了盒图,又称为 N - S 图。它是一种严格符合结构化程序设计原则的图形描述工具。

N - S 图的基本特点是通过矩形框描述模块内部程序的各个功能区域,并通过由外到内的矩形框嵌套表示程序的多层控制嵌套。

### (3) PAD 图

PAD 是问题分析图 (Problem Analysis Diagram) 的英文缩写,由日本日立公司首先推出,并得到了广泛的应用。它是符合结构化程序设计原则的图形描述工具。

PAD 图的基本特点是使用二维树形结构表示程序的控制流程,从上至下是程序进程方向,从左至右是程序控制嵌套关系。

### (4) PDL 语言

PDL 语言也称为伪码,或过程设计语言,它一般是某种高级语言稍加改造后的产物,可以使用普通的正文编辑软件或文字处理系统进行 PDL 的书写和编辑。

PDL 语言的语法规则分外部语法和内部语法。其中,外部语法用于定义程序中的控制结构 and 数据结构,内部语法则用于表示程序中的加工计算或条件。

### (5) 判定表

判定表是算法设计辅助工具,专门用于对复杂的条件组合关系及其对应的动作行为等给出更加清晰的说明,能够简洁而又无歧义地描述涉及条件判断的处理规则。

## 3. Jackson 程序设计方法

1983 年法国科学家 Jackson 提出了一种以软件中的数据结构为基本依据的程序算法设计方法。在以数据处理为主要内容的信息系统开发中,具有一定的应用价值。

Jackson 程序设计方法的基本设计途径是通过分析输入数据与输出数据的层次结构,由此对程序算法的层次结构进行推论。

为了方便由数据结构映射出程序结构,Jackson 将软件系统中所遇到的数据分为顺序、选择和重复三种结构,并使用图形方式加以表示。Jackson 程序结构也是顺序、选择和重复这三种结构,并可以使用与数据结构相同的图形符号表示。

## 4. 程序编码

在完成程序算法设计之后,接着需要编码。

### (1) 编程语言种类

- 低级语言:包括第一代机器语言与汇编语言,它们是直接面向机器的语言。
- 高级语言:指面向问题求解过程的语言,使用了与人的思维体系更加接近的概念和符号,一般不依赖于实现这种语言的计算机,具有较好的可移植性。

- 第四代语言 (4GL): 指一些面向问题的高级语言, 第四代语言是在更高级抽象的层次上表示数据与猜想结构, 它不需要规定程序算法细节。

## (2) 选择编程语言的依据

在对软件系统进行编码之前, 必须抉择使用什么样的程序设计语言实现这个软件系统。在选择编程语言时往往需要考虑诸多方面的因素, 例如软件项目的应用领域、软件问题的算法复杂性、软件的工作环境、软件在性能上的需要、软件中数据结构的复杂性、软件开发人员的知识水平和心理因素等。

## (3) 编程风格与质量

编程风格是编写程序时需要遵守的一些规则。在衡量程序质量时, 源程序代码的逻辑简明清晰、易读易懂是一个重要因素, 而这些都与编程风格有着直接的关系。

## (4) 影响程序工作效率的因素

一般说来, 程序工作效率会受到处理器计算速度、存储器存储容量和输入输出速度等几个方面因素的影响, 并与程序设计语言、操作系统、硬件环境等有着直接关系。因此, 在考虑程序工作效率时, 需要将诸多因素综合起来分析。

## 5. 程序算法复杂性度量

程序算法复杂性主要指模块内程序的复杂性。比较著名的程序算法复杂性度量方法是 McCabe 度量法, 其对程序复杂性的度量采用的是程序的环形复杂度, 计算公式是:

$$V(G) = m - n + p$$

其中,  $V(G)$  是程序有向图  $G$  中的环数,  $m$  是程序有向图  $G$  中的弧数,  $n$  是程序有向图  $G$  中的节点数,  $p$  是程序有向图  $G$  中分离部分的数目。

## 习 题

1. 试描述结构化程序的特点。

2. 某算法设计程序流程图如图 8-17 所示。试分析该算法为什么不能满足结构化程序设计的要求。为了使它满足结构化设计要求, 应该进行哪些方面的修改。

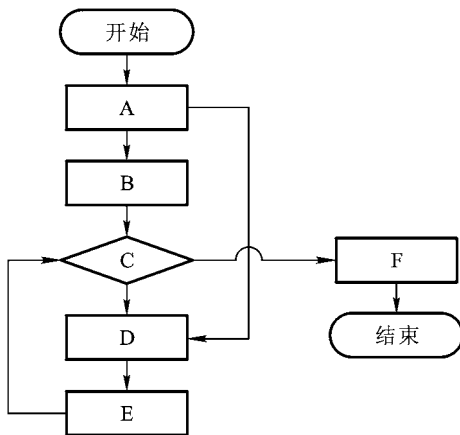


图 8-17 练习 2 程序流程图

3. 某算法设计程序流程图如图 8-18 所示。试将该图转换为 N - S 图、PAD 图、PDL 伪码, 并使用 McCabe 方法对该算法的复杂度进行估算。

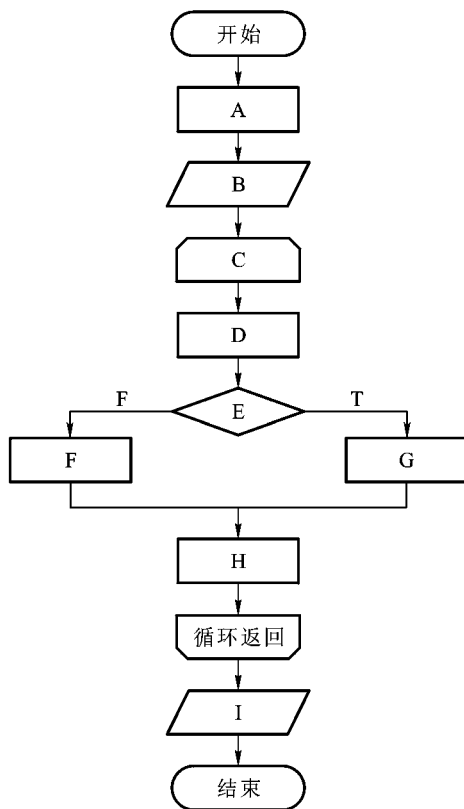


图 8-18 练习 3 程序流程图

4. 需要从 1 000 以内将能够被 7 整除的数查询出来, 并计算出这些数的和。试分别使用程序流程图、N - S 图、PAD 图和 PDL 伪码, 设计该程序问题的算法。然后使用 McCabe 方法对所设计的算法的复杂度进行估算。

5. 某零件数据库保存有零件进库情况的记录。其中, 零件按编号不重复地登记在数据库的“零件登记表”中, 而数据库中的“零件进库记录表”则以流水方式记录了每次零件进库时的编号、数量等信息。现要求按零件编号对零件进库情况进行汇总, 请使用 Jackson 程序设计方法设计该问题的程序算法。

6. 试举例说明软件项目中选择编程语言的依据。

## 第9章 软件测试

在开发软件的过程中，开发者的工作难免会出现差错，这些工作上的差错有可能在软件中遗留下错误或缺陷，并有可能影响软件今后的正常使用。应该说，软件错误是多方面的，例如规格定义错误、结构设计错误、算法设计错误、编码错误等。其中，越是早期错误，其对软件质量的影响越大。因此，软件错误应该尽早发现，并尽早克服。

软件测试是针对所开发软件的分析、设计与编码的最后一次复审，以获得对软件质量的再一次验证与确认。一般说来，开发前期分析、设计阶段产生的错误，首先需要通过严格的阶段性评审加以克服。但是，由于人认识上的局限性和软件错误的隐蔽性，有些错误可能在阶段性评审期间没有被发现，而且在程序编码过程中也有可能带来一些编码错误。实际情况是，假如软件错误在软件交付用户使用之前没有被发现并纠正，则这些错误迟早会在今后软件的工作过程中暴露出来，那时不仅改正这些错误的代价更高，而且往往会造成很恶劣的后果。因此，软件工程要求，在软件交付用户使用之前，应该对软件进行严格的全面的测试，尽可能地将软件中存在的错误发现出来，以确保软件质量。

### 9.1 软件测试基本概念

#### 9.1.1 测试目标

软件测试的目标是为了发现软件中隐藏的错误，应该说，这是一个非常清楚的目标。可是，由于受开发机构的利益、开发者个人的心理状态等诸多因素的影响，测试目标往往被人误解或误用。例如，一些软件开发机构可能希望通过测试而向用户证实软件中不存在错误，以此来表明软件能够完全满足用户的要求，于是开发机构在进行软件测试时，就可能产生回避软件错误的意图。他们往往会设计一些不易暴露软件错误的测试方案，选择那些难以使软件出故障的测试用例，或面对用户进行一些如同表演的所谓测试操作，等等。显然，这样的测试只会使软件错误被掩盖起来，而无益于软件质量的提高。

针对软件测试目标中存在的错误认识与做法，G.Myers 给出了关于软件测试目标的一些规则说明或定义，这些规则可以理解为以下几点：

- (1) 软件测试过程就是发现软件中错误的过程。
- (2) 应该力求设计出这样的测试方案，它极有可能发现迄今为止尚未发现的软件错误。
- (3) 评价一个测试是否成功，就是看这个测试是否发现了至今为止尚未发现的软件错误。

这些规则表明：测试只能用来发现程序中的错误，而不能用来证明程序的正确性。实际上，软件作为逻辑产品，其错误具有很大的隐蔽性，即使是经过了最严格的测试，软件中仍有可能还有没被发现的错误潜藏在程序中。

鉴于测试的目标是需要暴露程序中的错误，而又考虑到软件测试中个人的态度倾向、认知

倾向对测试可能带来干扰，一般认为，程序的编写者不适合于对自己编写的程序进行确认性测试（程序调试例外）。因此，在进行模块确认测试或软件综合测试的时候，通常需要由其他人员组成专门的测试小组来完成测试工作。

### 9.1.2 测试方法

软件测试中最基本的方法是黑盒测试法与白盒测试法，应该说任何软件产品都可以使用这两种方法进行测试。

#### 1. 黑盒测试法

黑盒测试是基于程序的外部功能规格而进行的测试。因此，黑盒测试又被称为功能测试。

黑盒测试法的基本特点如图 9-1 所示，它把有待测试的程序模块看作是一个黑盒子，并只对程序模块接口处的输入数据与输出数据进行测试。例如，检查程序模块的功能是否能够按照需求规格说明书的规定正常使用，是否能够按照设计要求正常地进行数据输入并能够正确地产生输出数据。而至于程序模块的内部结构与处理过程，黑盒测试时不需要对它们做任何考虑。

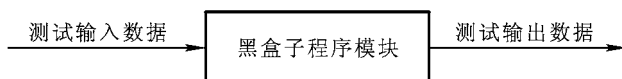


图 9-1 黑盒测试

黑盒测试一般以程序模块为单位进行，其适合于对程序模块的确认测试、系统集成测试和用户验收测试。

#### 2. 白盒测试法

白盒测试是基于程序的内部结构与处理过程而进行的测试。因此，白盒测试又被称为结构测试。

与黑盒测试相反，白盒测试的前提是可以把程序看成装在一个透明的白盒子里，测试者可以完全清楚地看到程序的内部结构与处理流程。正因此，白盒测试的内容也就是程序的内部算法细节，例如程序中的每一条语句、程序的执行路径、循环的操作次数、条件表达式、程序内部的数据结构等，它们是否都与算法设计说明保持一致。

在对程序进行低层单元测试，或进行程序调试，或进行程序错误定位时，往往需要用到白盒测试。

图 9-2 是白盒测试法的图示说明。

### 9.1.3 测试中的信息流

图 9-3 描述了软件测试过程中的信息流。从该图可以看到，测试过程中需要输入的信息包括：

（1）软件配置：包括需求规格说明书、软件设计说明书和源程序清单等。

（2）测试配置：包括测试计划、测试方案、测试用例等。实际上测试配置是软件配置的一个子集，最终交出的软件配置应该包括上述测试配置。

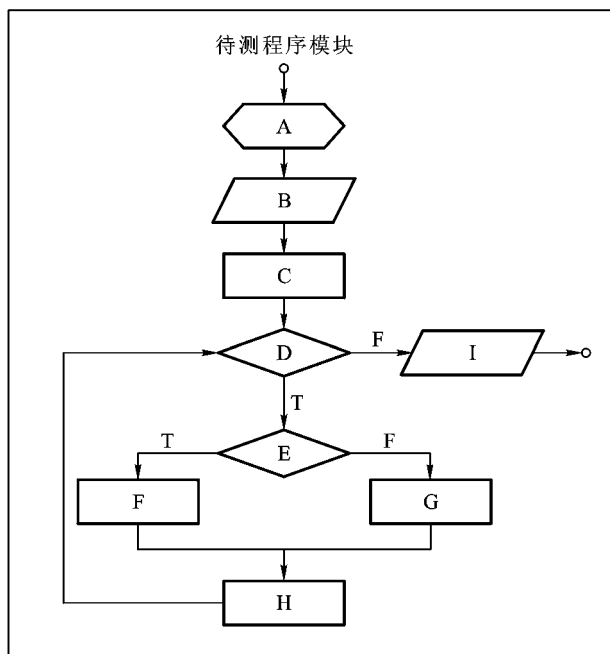


图 9-2 白盒测试

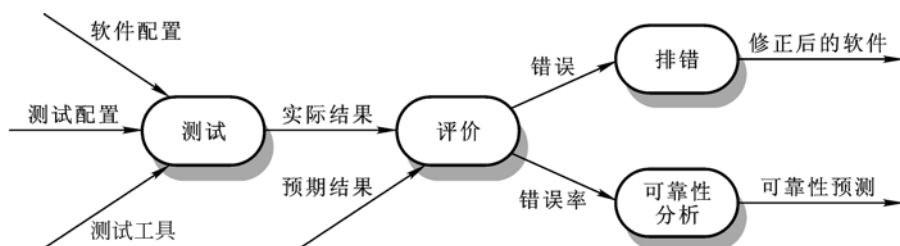


图 9-3 测试中的信息流

(3) 测试工具：可使用测试工具提高软件测试效率，例如，测试数据自动生成程序、测试分析程序、测试驱动程序等。

测试过程中还需要对测试结果进行评价。例如，比较测试得出的实际结果和预期的结果，如果两者不一致则很可能是程序中有错误。

在对测试结果进行收集和评价的时候，还需要对软件的可靠性进行预测。如果经常出现要求修改设计的严重错误，那么软件的质量和可靠性是值得怀疑的。反之，如果看起来软件功能完成得很正常，遇到的错误也很容易改正，则可以考虑以下两种可能：其一，软件的可靠性是可以接受的；其二，所进行的测试尚不足以发现严重的错误。但如果经过测试，一个错误也没有被发现，则有可能是因为测试配置不当，以致不能暴露软件中潜藏的错误。

针对测试过程中产生的结果，也可以用更形式化的方法进行评价。例如，通过测试时产生的错误率数据建立软件可靠性模型，由此可以估计软件将来出现错误的可能性，并进而对软件可靠性进行预测。



## 9.2 软件测试过程

与软件开发类似，软件测试也需要分阶段进行。但是，软件开发中的阶段是自顶向下的，从需求分析，到概要设计，再到详细设计；而软件测试中的阶段则与此刚好相反，首先是根据详细设计阶段制定的测试计划进行单元模块测试，然后根据概要设计阶段制定的测试计划进行系统集成测试，最后才是根据需求分析阶段制定的测试计划进行用户确认测试。

图 9-4 是对软件测试过程的图示说明。

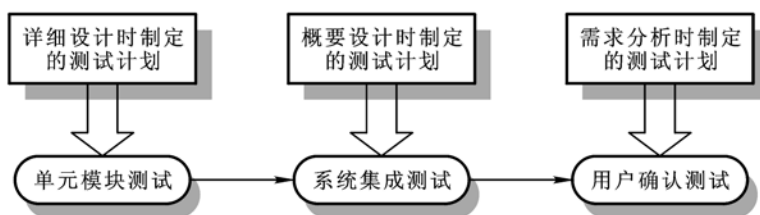


图 9-4 软件测试过程

### 9.2.1 单元测试

#### 1. 单元测试内容

单元测试的测试对象是单元模块，例如函数、过程，它们是组成系统的最小模块单位。在进行单元测试时，测试者应该通过详细设计说明书和源程序清单等，对模块的构造（内部数据结构、接口结构、内部程序结构等）有比较全面的认识。

单元测试一般以白盒测试为主，以黑盒测试为辅。并需要从以下几个方面对单元模块进行测试：

##### （1）模块接口测试

主要测试项目有：

- 调用所测模块时的输入参数与模块的形式参数在个数、属性、顺序上是否匹配。
- 所测模块调用子模块时，它输入给子模块的参数与子模块中的形式参数在个数、属性、顺序上是否匹配。
- 是否修改了只作输入用的形式参数。
- 输出给标准函数的参数在个数、属性、顺序上是否正确。
- 全局变量的定义在各模块中是否一致。

当模块需要通过外部设备进行输入输出操作时，则还需要测试以下项目：

- 文件属性是否正确。
- OPEN 语句与 CLOSE 语句是否正确。
- 规定的输入输出格式说明与输入输出语句是否匹配。
- 缓冲区容量与记录长度是否匹配。
- 在进行读写操作之前是否打开了文件。

- 在结束文件处理时是否关闭了文件。
- 是否有输入输出检测。

### (2) 局部数据结构测试

模块的局部数据结构是最常见的错误来源，需要进行以下方面的测试：

- 不正确或不一致的数据类型说明。
- 使用了尚未赋值或尚未初始化的变量。
- 错误的初始值或错误的缺省值。
- 变量名拼写或书写错误。
- 出现了数据类型冲突。

### (3) 路径测试

需要对模块中重要的执行路径进行测试。应当设计测试用例查找由于错误的计算、不正确的比较或不正常的控制流而导致的错误。

常见的不正确计算有：

- 运算的优先次序不正确或误解了运算的优先次序。
- 变量初始化不正确。
- 运算精度不够。
- 表达式的符号表示不正确。

常见的比较和控制流错误有：

- 出现了不同数据类型量之间的相互比较。
- 不正确地使用了逻辑运算符，或出现了优先次序错误。
- 因浮点数运算精度问题，两值不会相等，但需要进行相等比较。
- “差1”错，即不正确地多循环了一次或少循环了一次。
- 错误的或不可能的循环终止条件。
- 当遇到发散的迭代时不能终止的循环。
- 不适当地修改了循环变量。

### (4) 错误处理测试

比较完善的模块设计还应该对出错有所预见，并设置适当的出错处理，以便在程序出错时，能对出错程序重做安排，保证其逻辑上的正确性。应该对程序中的出错处理部分进行以下方面的测试：

- 对所出现的错误的描述难以理解；
- 出错的描述不足以对错误定位，不足以确定出错的原因；
- 显示的错误与实际的错误不符；
- 对错误的处理不正确；

### (5) 边界测试

在边界上出现错误是常见的。应该对以下方面的边界问题进行测试：

- 循环操作中的最大循环次数。
- 条件判断操作中的“>、>=、<、<=”等比较运算符。

## 2. 单元测试方法

单元测试通常在编码阶段进行。在源程序代码编制完成，经过评审和验证，确认没有语法错误之后，接着就可以进行单元测试。

模块并不是一个完全独立的程序，在进行模块测试时，需要考虑它与外界的联系，并需要使用一些辅助模块去模拟与被测模块相联系的其他模块。

单元测试中需要使用辅助模块分为以下两种：

(1) 驱动模块：相当于被测模块的主程序。可以通过它调用被测模块，接收测试数据，然后由它把这些测试数据传送给被测模块，最后再输出实测结果。

(2) 桩模块：也叫做存根模块，用以代替被测模块需要调用的子模块。桩模块仅仅需要做很少量的数据操作，例如，显示一个提示，以表明自己已被调用。

实际上，单元测试时，测试模块与驱动模块、桩模块，共同构成了一个“单元测试环境”，如图 9-5 所示。

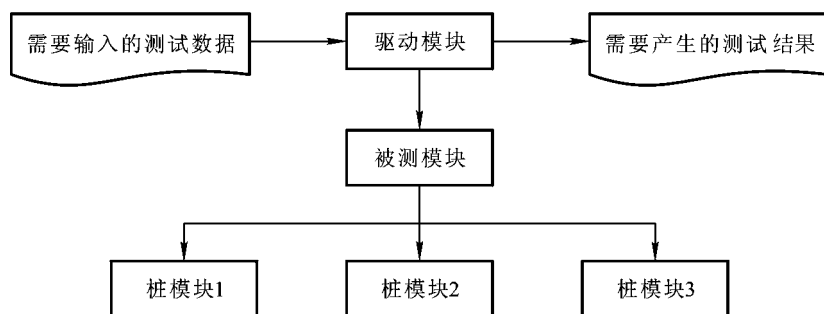


图 9-5 单元测试环境

## 9.2.2 集成测试

### 1. 集成测试任务

系统集成测试一般要求由专门的测试小组承担测试任务。

通常，系统集成测试兼有对系统进行组装与检测双重任务。一方面，系统集成测试需要将经过测试的模块组织起来，由此将分散的模块装配成一个统一的系统。另一方面，在进行系统装配的过程中，又还要对每个需要集成到系统之中的模块做进一步的检测，从中发现将要组织成为系统的模块是否能够按照软件的结构设计要求进行组装，诸多模块之间是否具有良好的协作性，它们是否能够协同工作。

具体说来，集成测试涉及以下方面的测试任务：

- (1) 在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；
- (2) 一个模块的功能是否会对另一个模块的功能产生不利的影响；
- (3) 各个子功能组合起来，能否达到预期要求的协作功能；
- (4) 全局数据结构是否有问题；
- (5) 单个模块的计算误差累积起来，是否会放大进而达到不能接受的程度。

## 2. 非渐增组装测试

系统集成时主要有非渐增组装测试和渐增组装测试这两种方法。其中，非渐增组装测试是一种一次性地进行系统组装的方法。这种方法要求先完成单元模块的确认测试，然后把所有模块按设计要求放在一起组合成系统。

由于非渐增组装测试将单元模块的确认测试与系统集成测试分开进行，诸多模块的确认测试工作可以并行开展，能够非常方便地将各项测试任务分配到各个不同的测试小组或个人。因此，非渐增组装测试能够充分利用人力，并能够加快测试速度。

但非渐增组装测试需要分别单独测试每个模块，因此需要编写更多的测试程序，例如驱动模块、桩模块，由此会使测试工作量加大。另外，非渐增式测试一下子把所有模块组合在一起，如果发现错误则较难诊断定位。此外，非渐增式测试也不利于模块接口错误的尽早发现。

## 3. 渐增组装测试

渐增组装测试则是一种将单元模块的确认测试与集成测试结合在一起的测试方法。这种方法的特点是，分别把需要集成到系统中的模块按照一定的次序，逐个集成到系统中，并在进行模块之间协作性测试的同时对模块的功能进行确认测试。

渐增组装测试具有多方面的优点。

其一，渐增组装利用已测试过的模块作为部分测试软件，因此可以减少一部分测试工作量。

其二，渐增组装可以较早发现模块间的接口错误。

其三，在使用渐增式测试方法时，如果发生错误则往往和最近加进来的那个模块有关，因此便于错误诊断与定位。

其四，渐增组装把已经测试好的模块和新加进来的那个模块一起测试，已测试过的模块可以在新的条件下受到新的检验，因此这种方法对程序的测试更彻底。

应该说，渐增组装测试比非渐增组装测试具有更大的优越性。但是，渐增组装将单元模块的确认测试与集成测试结合到了一起，因此不便于测试任务的分配，比起非渐增组装来，渐增组装方法需要更长的测试时间。

当使用渐增方式进行系统集成时，有自顶向下和自底向上两种方法。下面分别介绍这两种方法。

### (1) 自顶向下渐增集成

自顶向下渐增是最常使用的系统集成。其特点是从主控制模块开始，沿着软件的控制层次向下移动，从而逐个地把各个模块集成到系统中来。

由于自顶向下渐增集成是最高层的主控制模块开始的，在测试模块时，被测模块的上级模块已经测试通过了，因此被测模块不需要“驱动模块”。但是，被测模块的下级模块还没有集成进来，因此被测模块需要“桩模块”，如图 9-6 所示。

按自顶向下渐增集成时，还需要考虑模块的组装顺序。可以按照系统的层次结构，采用深度优先或层次优先进行模块组装。其中，深度优先是按系统的某个控制通路安排组装顺序，其往往体现为一个比较完整的子系统或功能块。而层次优先则是体现为将同一个控制层次上的所有模块按顺序组装起来。相比之下，深度优先具有更多的合理性，例如，可以使开发者更早地感受到一项完整功能的实现所带来的成功体验。

可以按以下步骤进行自顶向下渐增集成：

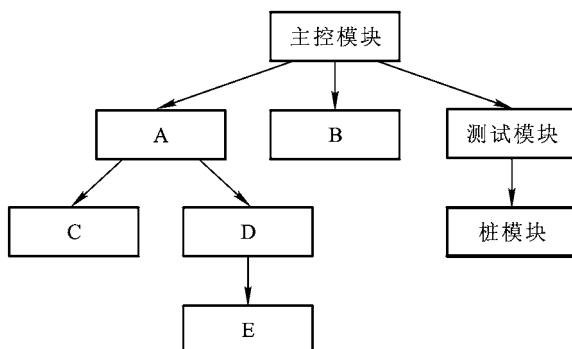


图 9-6 自顶向下渐增集成

第一步，对主控制模块进行测试，测试时用“桩模块”代替所有直接附属于主控制模块的模块。

第二步，根据选定的优先策略（深度优先或层次优先），每次用一个实际模块代换一个“桩模块”，并为这个新结合进来的模块准备新的“桩模块”。

第三步，对新结合进来的模块进行测试。

第四步，为了保证加入模块没有引进新的错误，可能需要进行回归测试（即全部或部分地重复以前做过的测试）。

从第二步开始不断地重复进行上述步骤，直到将所有模块都集成到系统之中为止。

### （2）自底向上渐增集成

自底向上渐增集成是从软件结构最低层的模块开始进行组装。由于自底向上渐增是从底部向上结合模块，在测试模块时，它的下级模块已经测试通过了，因此这种方式不需要“桩模块”。但是，被测模块的上级模块还没有集成进来，因此需要为正在测试的模块准备“驱动模块”。

可以按照以下步骤进行自底向上渐增集成：

第一步，通过“驱动模块”对最底层模块进行并行测试。

第二步，用实际模块代替驱动模块，与它已测试的直属子模块组装成子系统。

第三步，为子系统配备驱动模块，进行新的测试。

第四步，从第二步开始不断地重复进行上述步骤，直到将所有模块都集成到系统之中（已将主模块集成进来）为止。

### （3）集成策略比较

上面介绍了自顶向下和自底向上两种渐增策略。

自顶向下渐增的优点是不需要“驱动模块”，能够在测试阶段的早期实现并验证系统的主要功能，而且能在早期发现上层模块的接口错误。但这种方法需要“桩模块”，底层关键模块中的错误发现较晚，并且不便于测试工作的并行开展。

自底向上渐增则与自顶向下渐增刚好相反，需要“驱动模块”，但不需要“桩模块”。因此，自底向上渐增不便于在早期发现上层模块的接口错误，但底层关键模块中的错误能够被较早发现，并且早期测试工作能够并行开展，有利于加快测试速度。

在测试实际的软件系统时,应该根据软件的特点以及工程进度安排,选用适当的测试策略。许多情况下,可以将自顶向下与自底向上结合起来使用。例如,基本上使用自顶向下方法进行集成测试,但在测试早期,为了便于测试工作并行开展,可以使用自底向上的方法对软件中的少数关键模块进行专门测试。

### 9.2.3 确认测试

确认测试又称有效性测试或验收测试。它的任务是验证软件的功能、性能以及其他特性等,是否与用户的要求保持一致,并得到用户确认。

实际上,软件的功能要求与性能要求在软件需求规格说明中已经明确规定了,在软件需求规格说明书中描述了全部用户可见的软件属性,其中有一节叫做有效性准则,它包含的信息就是软件确认测试的基础。因此,确认测试也就是按照软件需求规格说明书的要求进行验证。

图 9-7 是确认测试阶段的工作流程图。其中,首先需要进行的是软件有效性测试以及软件配置复审,接着需要进行的是验收测试和安装测试。只有在通过了专家鉴定之后,确认测试工作才能结束,所开发的软件才能成为可交付使用的软件。

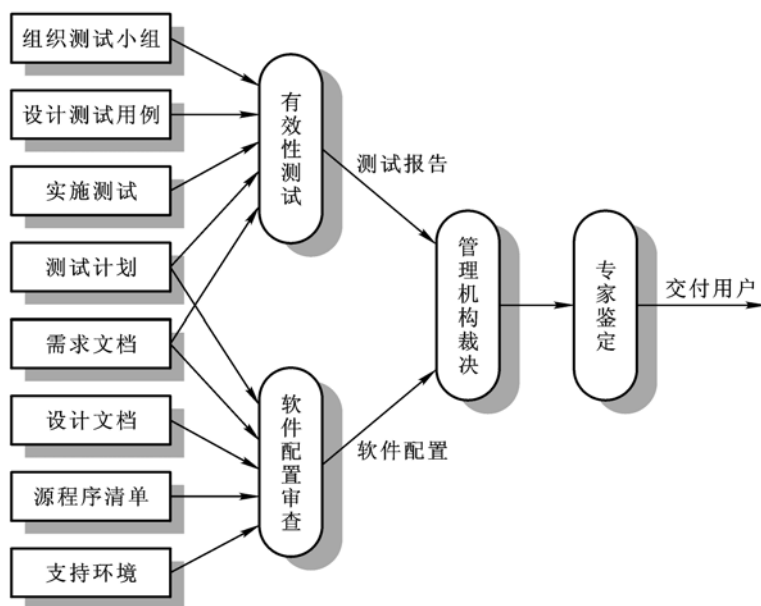


图 9-7 确认测试工作流程

#### (1) 进行软件有效性测试

有效性测试是在模拟的环境(可能就是开发的环境)下进行,需要使用黑盒测试方法对软件进行有效性验证,用于确定所开发的软件是否能够满足需求规格说明书所规定的需求。为此,需要首先制定测试计划,确定测试种类,提出测试方案,描述具体的测试用例等。通过实施预定的测试计划和测试方案,确定软件的特性是否与需求相符,以确保所有的软件功能需求都能得到满足,所有的软件性能需求都能达到,所有的文档都是正确且便于使用。同时,对其他软件需求,例如可移植性、兼容性、出错自动恢复、可维护性等,也都要进行测试,确认是否满

足要求。

在全部软件测试的测试用例运行完毕之后，所有的测试结果可以分为以下两类：

第一类：测试结果与预期的结果相符。这说明软件的这部分功能或性能特征与需求规格说明书相符合，从而接受这部分程序。

第二类：测试结果与预期的结果不符。这说明软件的这部分功能或性能特征与需求规格说明不一致，因此要为其提交一份问题报告。

### （2）进行软件配置复查

软件配置复查的目的是保证软件配置的所有成分都齐全，各方面的质量都符合要求，具有维护阶段所必需的细节，而且已经编排好分类的目录。

除了按合同规定的内容和要求，由人工审查软件配置之外，在确认测试的过程中，应当严格遵守用户手册和操作手册中规定的使用步骤，以便检查这些文档资料的完整性和正确性。必须仔细记录发现的问题，包括遗漏与错误，并进行适当地补充与修正。

### （3）Alpha 测试

Alpha 测试是指在开发环境下由用户进行的测试，软件是在一个自然设置状态下被使用。担任测试人员的用户是除开软件开发人员之外首先见到软件产品的人，他们提出的修改意见具有特别的价值。

进行 Alpha 测试时，开发者往往就坐在用户旁边，随时记下错误情况和使用中的问题。显然，这是一种在受到控制的环境下进行的测试。Alpha 测试的目的是让用户就软件的功能、操作界面、性能、可使用性、可靠性和支持等，进行直截了当的评价。

Alpha 测试可以从软件产品编码结束之时开始，或在模块测试完成之后开始，也可以是在确认测试过程中产品达到一定的稳定和可靠程度之后再开始。为了使 Alpha 测试能够顺利进行，应该为 Alpha 测试准备好有关的操作手册，以供用户进行测试操作时使用。

### （4）Beta 测试

Beta 测试则是由软件用户在软件实际使用环境下进行的测试。与 Alpha 测试不同的是，开发者通常不在测试现场。因此，Beta 测试是在开发者无法控制的环境下进行的软件现场应用。在 Beta 测试中，由用户自己记录下遇到的所有问题，包括真实的以及主观认定的，并需要将有关问题的记录以及意见报告给开发者，开发者则在综合用户的报告之后，做出修改。在经过 Beta 测试之后，软件产品可交付给全体用户使用。

Beta 测试的重点测试内容是软件产品的支持性，涉及用户文档、客户培训等。Beta 测试要求在 Alpha 测试之后进行。只有当 Alpha 测试达到一定的可靠程度时，才能开始 Beta 测试。由于它处在整个测试的最后阶段，不能指望这时发现主要问题。同时，产品的所有手册文本等也应该在此阶段完全定稿。

### （5）确认测试结果

确认测试结果可能是以下两种情况之一：

其一，功能和性能与用户的要求一致，软件可以接受。

其二，功能和性能与用户的要求有差距。

若是后一种情况，则通常是软件需求分析出现了偏差。这时需要将软件的各项缺陷与问题以报告形式向用户提出，然后通过与用户进行协商，找到解决问题的办法。

确认测试应交付的文档有：确认测试分析报告、最终的用户手册和操作手册、项目开发总结报告。

## 9.3 软件测试用例设计

设计测试用例就是为测试准备测试数据，是测试阶段的关键技术问题，直接影响测试效果的好坏。

实际上，测试用例不同，发现程序错误的能力也就不同，为了提高测试效率降低测试成本，应该选用高效的测试用例。

测试用例设计的基本目标是，确定一组最可能发现某个错误或某类错误的测试数据。已经研究出许多设计测试数据的技术，这些技术各有优缺点，没有哪一种是最好的，更没有哪一种可以代管其余所有技术；同一种技术在不同的应用场合效果可能相差很大，因此，通常需要联合使用多种设计测试用例的技术。其中，最主要的设计用例的技术有：适用于黑盒测试的等价划分、边界值分析以及错误推测法等；适用于白盒测试的逻辑覆盖法。通常的做法是，用黑盒法设计基本的测试方案，再用白盒法补充一些方案。

实际上，没有一种测试方法能代替所有其他的测试方法。此外，不同方法各有所长，用一种方法设计出的测试用例可能最容易发现某些类型的错误，对另外一些类型的错误可能不易发现。因此，在软件系统进行实际测试时，往往联合使用多种不同的设计方法进行测试用例的设计，由此而形成一种综合的测试策略。通常的做法是，用黑盒法设计基本的测试方案，再用白盒法补充一些必要的测试方案。具体地说，可以使用下述策略结合各种方法：

(1) 在任何情况下都应该使用边界值分析的方法。经验表明，用这种方法设计出的测试用例暴露程序错误的能力最强。

(2) 必要时用等价划分法补充测试用例。

(3) 必要时再用错误推测法补充测试用例。

(4) 对照程序逻辑，检查已经设计出的测试方案。可以根据对程序可靠性的要求采用不同的逻辑覆盖标准。

应该说明的是，即使使用上述综合策略设计测试方案，仍然不能保证测试将发现一切程序错误；但是，这个策略则能够使测试成本与测试效果之间有一个比较合理的平衡。

### 9.3.1 白盒测试用例设计

白盒测试用例设计主要采用的是逻辑覆盖，这是一种以程序内部逻辑结构为依据的用例设计方法，包括语句覆盖、判定覆盖、条件覆盖、判定—条件覆盖、条件组合覆盖和路径覆盖等几种覆盖强度各不相同的逻辑覆盖形式。

#### 1. 语句覆盖

为了暴露程序中的错误，至少每个语句应该执行一次。语句覆盖的含义是，选择足够多的测试数据，使被测程序中每个语句至少执行一次。

例如，图 9-8 中的程序流程图。它的源程序如下：



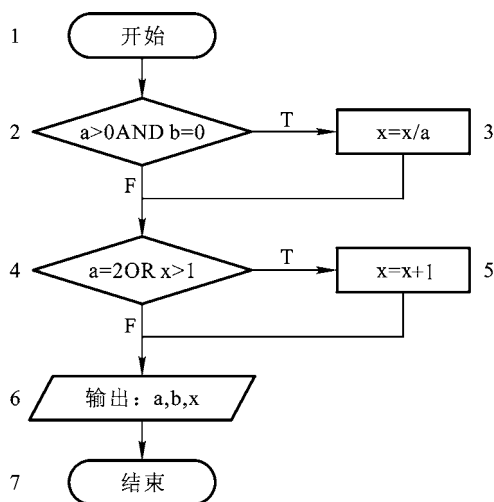


图 9-8 被测试模块的程序流程图

```

void example ( int a, int b, int x )
{
    if ( a > 1) && ( b == 0)
        x = x / a;
    if ( a == 2) || ( x > 1)
        x = x + 1;
    printf(a,b,x);
}

```

为了使程序中每个语句都执行一次,以达到语句覆盖的目标。需要准备以下测试数据 :a=2, b=0, x=4, 其执行路径是: 1-2-3-4-5-6-7。

应该说, 语句覆盖是一种弱覆盖标准, 对程序的逻辑覆盖很少, 在上面例子中两个判定条件都只测试了条件为真的情况, 如果条件为假时处理有错误, 显然不能发现。

## 2. 判定覆盖

判定覆盖含义是: 不仅每个语句必须至少执行一次, 而且每个判定的每种可能的结果都应该至少执行一次。

对于图 9-8 中的程序, 即是要求: (a > 1) && (b == 0) 与 (a == 2) || (x > 1) 这两个判断表达式各出现 “T”、“F” 结果至少一次。

下面两组测试数据就可做到判定覆盖:

(1) a = 3, b = 0, x = 3 (执行路径: 1-2-3-4-6-7) (判断式: T, F)

(2) a = 2, b = 1, x = 1 (执行路径: 1-2-4-5-6-7) (判断式: F, T)

判定覆盖能够包含语句覆盖, 因此判定覆盖比语句覆盖强, 但判定覆盖对程序逻辑的覆盖程度仍然不高。

## 3. 条件覆盖

条件覆盖的含义是: 不仅每个语句至少执行一次, 而且使判定表达式中的每个条件都取到

各种可能的结果。

对于图 9-8 中的程序，即是要求：(a >1)、(b == 0)、(a ==2)、(x >1)这四个条件表达式各出现“T”、“F”结果至少一次。

下面两组测试数据就可以达到条件覆盖标准：

(1) a=2, b=0, x=4 (执行路径：1-2-3-4-5-6-7) (条件式：T, T, T, T)

(2) a=1, b=1, x=1 (执行路径：1-2-4-5-6-7) (条件式：F, F, F, F)

条件覆盖通常比判定覆盖强，因为它使判定表达式中每个条件都取到了两个不同的结果，判定覆盖却只关心整个判定表达式的值。例如，上面两组测试数据也同时满足判定覆盖标准。但是，也可能有相反的情况：虽然每个条件都取到了两个不同的结果，判定表达式却始终只取一个值。这也就是说，条件覆盖并不一定能够包含判定覆盖。

#### 4. 判定-条件覆盖

判定-条件覆盖是指既能满足判定覆盖，又能满足条件覆盖，其含义是：选取足够多的测试数据，使得判定表达式中的每个条件都取到各种可能的值，而且每个判定表达式也都取到各种可能的结果。

对于图 9-8 中的程序，下述两组测试数据能够满足判定-条件覆盖标准：

(1) a=2, b=0, x=4 (执行路径：1-2-3-4-5-6-7)

(2) a=1, b=1, x=1 (执行路径：1-2-4-6-7)

#### 5. 条件组合覆盖

条件组合覆盖是更强的逻辑覆盖标准，它要求选取足够多的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。

对于图 9-8 中的程序，它有两个判断式，每个判断式又含两个条件式，因此可以产生八种可能的条件组合，而下面的四组测试数据可以使这八种条件组合中的每种至少出现一次：

(1) a=2, b=0, x=4 (执行路径：1-2-3-4-5-6-7) (条件式：[T, T], [T, T])

(2) a=2, b=1, x=1 (执行路径：1-2-4-5-6-7) (条件式：[T, F], [T, F])

(3) a=1, b=0, x=2 (执行路径：1-2-4-5-6-7) (条件式：[F, T], [F, T])

(4) a=1, b=1, x=1 (执行路径：1-2-4-6-7) (条件式：[F, F], [F, F])

显然，满足条件组合覆盖标准的测试数据，也一定满足判定覆盖、条件覆盖和判定—条件覆盖标准。但是，满足条件组合覆盖标准的测试数据并不一定能使程序中的每条路径都执行到，例如，上述四组测试数据都没有测试到路径：1-2-3-4-6-7。

#### 6. 点覆盖

点覆盖的含义是：选取足够多的测试数据，使得程序执行路径至少经过了程序图中每个节点一次。显然，点覆盖标准和语句覆盖标准是相同的。

#### 7. 边覆盖

边覆盖的含义是：选取足够多的测试数据，使得程序执行路径至少经过程序图中每条边一次。通常，边覆盖和判定覆盖是一致的。

#### 8. 路径覆盖

路径覆盖的含义是：选取足够多的测试数据，使程序的每条可能的路径都至少执行一次（如果程序图中有环，则要求每个环至少经过一次）。

对于图 9-8 中的程序，下面四组测试数据可以满足路径覆盖的要求：

- (1)  $a=1, b=1, x=1$  (执行路径：1-2-4-6-7)
- (2)  $a=1, b=1, x=2$  (执行路径：1-2-4-5-6-7)
- (3)  $a=3, b=0, x=1$  (执行路径：1-2-3-4-6-7)
- (4)  $a=2, b=0, x=4$  (执行路径：1-2-3-4-5-6-7)

路径覆盖相当于判定组合覆盖，是一种相当强的逻辑覆盖标准，它保证程序中每条可能的路径都至少执行一次。但是，为了做到路径覆盖只需考虑每个判定表达式的取值，并没有检验表达式中条件的各种可能组合情况。如果把路径覆盖和条件组合覆盖结合起来，则可以设计出检错能力更强的测试数据。

### 9.3.2 黑盒测试用例设计

#### 1. 等价类划分

等价类划分是黑盒测试时经常采用的用例设计技术。

等价类划分的特点是把所有可能的输入数据（有效的和无效的）划分成若干个等价类，并做出如下假定：每个等价类中的一个典型值在测试中的作用与这一类中所有其他值的作用相同。因此，可以从每个等价类中只取一组数据作为测试数据。

在使用等价类划分法设计测试用例时，首先需要划分输入数据的等价类，为此需要研究程序的功能说明，从而确定输入数据的有效等价类和无效等价类。

在确定输入数据的等价类时，常常还需要分析输出数据的等价类，以便根据输出数据的等价类导出对应的输入数据的等价类。

例如，学生成绩的录入与输出。假如录入的是百分制成绩，范围是：0—100 分；输出的是等级制成绩，等级是：优秀（85—100 分）、合格（60—84 分）、不合格（0—59 分）。则根据等价划分的原则，可以考虑以下的用例设计方案。

根据录入数据，可以确定以下 3 个等价类：

- (1) 录入数据  $< 0$  (无效类)
- (2) 录入数据  $\geq 0$  并且 录入数据  $\leq 100$  (有效类)
- (3) 录入数据  $> 100$  (无效类)

根据输出结果，还可以将录入数据的有效类继续细分为以下 3 个等价类：

- (1) 优秀：录入数据  $\geq 85$  并且 录入数据  $\leq 100$  (有效类)
- (2) 合格：录入数据  $\geq 60$  并且 录入数据  $< 85$  (有效类)
- (3) 不合格：录入数据  $\geq 0$  并且 录入数据  $< 60$  (有效类)

因此，根据对录入数据与输出结果的综合考虑，可以从上述 5 个等价类中各取一个数据组成一组测试用例，如：-20, 20, 80, 90, 120

划分等价类往往需要经验，下面几条规则可能有助于等价类的划分：

(1) 如果规定了输入值的范围，则可划分出一个有效的等价类（输入值在此范围内），两个无效的等价类（输入值小于最小值和大于最大值）。

(2) 如果规定了输入数据的个数，则类似地也可以划分出一个有效的等价类和两个无效的等价类。

(3) 如果规定了输入数据的一组值, 而且程序对不同输入值做不同处理, 则每个允许的输入值是一个有效的等价类, 此外还有一个无效的等价类(任一个不允许的输入值)。

(4) 如果规定了输入数据必须遵循的规则, 则可以划分出一个有效的等价类(符合规则)和若干个无效的等价类(从各种不同角度违反规则)。

(5) 如果规定了输入数据为整型, 则可以划分出正整数、零和负整数等三个有效类; 如果程序的处理对象是表格, 则应该使用空表以及含一项或多项的表。

在划分出等价类以后, 接着可以按以下步骤进行用例设计:

(1) 设计一个新的测试用例, 以尽可能多地覆盖尚未被覆盖的有效等价类, 重复这一步骤直到所有有效等价类都被覆盖为止。

(2) 设计一个新的测试用例, 使它覆盖一个而且只覆盖一个尚未被覆盖的无效等价类, 重复这一步骤直到所有无效等价类都被覆盖为止。(注意: 通常程序发现一类错误后就不再检查是否还有其他错误, 因此, 应该使每个测试用例只覆盖一个无效的等价类。)

## 2. 边界值分析

经验表明, 程序在处理边界问题时最容易发生错误。例如, 数组下标、条件判断、循环的边界处可能出现遗漏或错误。因此, 需要在边界处设计专门的测试用例, 用于发现程序运行在边界附近时, 是否会发生错误。

例如前面的学生成绩的录入与输出问题。其在 0、60、85、100 这些边界点上最容易出错。如下面的语句:

```
if (grade >= 85) && (grade <= 100)
    printf("优秀");
```

其中, 就有可能因为编码疏忽, 而使条件式 `grade >= 85` 漏了一个 "=" 号, 由此使得 85 分不能输出结果。

在使用边界值分析方法设计测试用例时, 首先需要确定边界情况, 然后在边界附近选取测试数据, 应该选取刚好等于、稍小于和稍大于等价类边界值的数据作为测试数据。

例如学生成绩的录入与输出问题。可以按照边界值分析原则设计出以下一组测试用例。即:  $(-1, 0, 1), (59, 60, 61), (84, 85, 86), (99, 100, 101)$ 。

## 3. 错误推测

不同类型不同特点的程序通常有一些特别容易出错的地方, 具有丰富测试经验的测试人员往往能够很直接地找出这些错误来。

错误推测法依靠的就是测试人员的测试经验与直觉。它的基本想法是列举出程序中可能有错误和容易发生错误的特殊情况, 并且根据它们选择测试用例。例如前面单元测试中列出的一些常见错误, 就是模块测试经验的总结。

对于程序中容易出错的情况, 也可以总结出一些经验来。例如, 输入数据为零或输出数据为零往往容易发生错误; 在进行数据表操作时, 一个没有任何记录的空表容易带来操作错误; 等等。

此外, 测试经验还表明: 程序中的错误还具有成堆聚集的特点, 也就是说, 一段程序中已经发现的错误数目往往和尚未发现的错误数成正比。这可能与设计、编码时的工作疏忽或理解错误等因素有关。因此, 一旦在程序的某个位置发现了错误, 则意味着需要对这段程序进行重

点测试，应该着重测试那些已发现了较多错误的程序段。

## 9.4 面向对象测试

对于传统的软件系统来说，单元测试集中测试最小的可编译的程序单元，一旦把这些单元都测试完之后，就把它集成到程序结构中去，并进行集成测试。在系统集成成功实现之后，最后还需要进行确认测试，以发现软件需求中的错误。应该说，面向对象测试有着与此相似的过程，但测试内容则略有不同。

### 9.4.1 面向对象单元测试

当考虑面向对象单元测试时，单元的概念发生了改变。“封装”导致了类和对象成为了最小的可测试单元，但一个类可以包含一组不同的操作，而一个特定的操作又可能被多个子类继承而共同拥有。因此，对于面向对象的软件来说，单元测试的含义发生了很大变化。

实际上，在进行面向对象单元测试时，已经不能再孤立地测试单个操作，而应该把操作作为类的一部分来测试。例如，在A类中定义了一个叫做op的操作，假如op被A类的多个子类继承，则每个子类都可以使用操作op。但是，A类的不同的子类调用op时所处理的是它自己的私有属性，而不同的子类有不同属性集。因此，需要以类为单位，在每个子类中分别测试op。

### 9.4.2 面向对象集成测试

面向对象软件是基于类而构造的，类之间的关系是依赖与引用关系。因此，面向对象软件中已经没有了分层的控制结构，传统的自顶向下和自底向上的集成策略也就没有意义了。此外，由于构成类的成分彼此间存在直接或间接的交互，采用传统的渐增组装方式，一次集成一个操作到类中，通常也就不可能了。

通常情况下，面向对象软件的集成测试可以采用以下两种策略。

(1) 基于线程的测试：其特点是把响应系统的一个输入或一个事件所需要的一组类集成起来。分别集成并测试每个线程，同时应用回归测试以保证没有产生副作用。

(2) 基于使用的测试：其特点是首先测试几乎不使用其他类的那些类，称为独立类。在把独立类都测试完之后，接下来测试使用独立类的下一个层次的类，称为依赖类。对依赖类的测试一个层次一个层次地持续进行下去，直至构造出整个软件系统为止。

### 9.4.3 面向对象确认测试

与传统的确认测试一样，面向对象确认测试也是面向用户的，其主要测试内容是用户可见的动作和用户可识别的输出。因此，在进行面向对象确认测试时，已经不需要考虑类的内部构造以及类之间相互连接等诸多设计细节了。

在进行用户确认测试时，测试人员需要研究系统的用例模型和活动模型，并需要以此为依据，设计出确认测试时的用户操作脚本。

## 9.5 软件调试

软件测试的目的是尽可能多地发现软件中的错误,但是,在发现错误之后还需要改正错误,因此需要对软件进行调试。

软件调试也叫做排错,涉及以下两个步骤。

(1) 诊断:用于确定程序中出现错误的性质与错误位置。

(2) 排错:对出现错误的程序段进行修改,由此排除错误。

在进行软件调试的上述两个步骤中,诊断是关键。应该说,当程序中出现错误的性质与错误位置被诊断出来以后,改错只是一件相对简单的工作。

### 9.5.1 调试方法

#### 1. 输出存储器内容

这种方法通常以八进制或十六进制的形式输出存储器的内容。这是一种效率很低的调试方法,其主要缺点是:

(1) 很难把存储单元和源程序变量对应起来。

(2) 输出信息量极大,而且大部分是无用的信息。

(3) 输出的是程序的静态映像,然而为了找出故障,往往需要研究程序的动态行为。

(4) 输出的存储器内容常常并不是程序出错时的状态,使其往往不能提供有用的调试线索。

(5) 输出信息的形式不易阅读和解释。

#### 2. 在程序中插入输出语句

这种方法是把程序设计语言提供的一些输出语句插到源程序的关键位置中去,以便输出关键变量的中间值。这种方法能够对源程序的执行情况进行一定的动态追踪,能够显示程序的动态行为,而且给出的信息容易和源程序对应起来。

这种方法的缺点主要是:

(1) 可能会输出大量的需要分析的信息,对于大型系统来说,情况更是如此。

(2) 必须修改源程序才能插入输出语句,这有可能改变一些关键的时间关系,从而可能掩盖错误。

#### 3. 使用自动调试工具

使用自动调试工具进行程序调试是目前使用最多的调试方法。在一些集成开发环境中,自动调试工具往往和整个程序创建工作结合在一起,因此可以非常有效地提高程序调试速度与质量。

自动调试工具一般具有对程序的动态调试功能,能够发现程序运行过程中出现的错误。自动调试工具主要的调试功能包括逐行或逐过程地执行源程序、在源程序中设置中断点、在源程序中设置需要监控的变量或表达式等。

### 9.5.2 调试策略

调试策略是指在对程序进行调试时可以采取的对策。常用的调试策略有:

### 1. 试探法

调试人员分析错误征兆, 猜测故障的大致位置, 然后使用前述的一两种调试方法检测程序中被怀疑位置附近的信息, 由此获得对程序错误的准确定位。

### 2. 回溯法

调试人员分析错误征兆, 确定最先发现“症状”的位置, 然后人工沿程序的控制流程往回追踪源程序代码, 直到找出错误根源或确定故障范围为止。

回溯法对于小程序而言是一种比较好的调试策略, 往往能把故障范围缩小为程序中的一小段代码, 仔细分析这段代码不难确定故障的准确位置。但是对于大规模的程序, 由于需要回溯的路径数目太多, 以致回溯变得困难起来。

### 3. 对分查找法

如果已经知道每个变量在程序内若干个关键点的正确值, 则可以用赋值语句或输入语句在程序中点附近“注入”这些变量的正确值, 然后检查程序的输出。如果输出结果是正确的, 则故障在程序的前半部分; 反之, 故障在程序的后半部分。对于程序中有故障的那部分再重复使用这个方法, 直到把故障范围缩小到容易诊断的程度为止。

### 4. 归纳法

归纳法是一种从个别推断一般的系统化的错误定位方法。归纳法往往以程序的错误征兆为线索, 通过分析这些线索之间的关系, 由此找出故障。

归纳法的使用主要有下述四个步骤:

#### (1) 收集有关的数据

列出已经知道的关于程序工作的一切数据, 然后分析哪些数据结果是正确的, 哪些数据结果是错误的。

#### (2) 组织数据

因为归纳法是从特殊推断一般的方法, 所以需要整理数据以便发现规律。常用的组织数据的方法是对数据进行分类。

#### (3) 导出假设

分析线索之间的关系, 力求找出它们的规律, 从而提出关于故障的一个或多个假设。如果无法做出推测, 则可以再设计并执行一些测试用例, 以便获得更多的数据; 如果可以做出多种假设, 则可以首先选取最有可能成为出错原因的那一个假设。

#### (4) 证明假设

假设不等于事实, 不经证明就根据假设排除故障, 往往只能消除错误的征兆或只能改正部分错误。因此, 需要对假设的合理性证明。证明假设的方法是, 用它解释所有原始的测试结果。如果能圆满地解释一切现象, 则假设得到证实, 否则要么是假设不成立或不完备, 要么是有多故障同时存在。

### 5. 演绎法

演绎法则从一般原理或前提出发, 经过排除和精化的过程推导出结论。使用演绎法进行程序调试时, 需要先列出所有看来可能成立的原因或假设, 然后一个一个地排除列举出的原因, 最后, 证明剩下的原因确实是错误的根源。

演绎法的使用主要有下述四个步骤:

### (1) 列出所有可能的原因

根据错误信息，将所有可能的原因以假设形式列举出来，对于这些假设，这时并不需要对它们作出任何解释，而只是将它们作为可能的原因提出来。

### (2) 用已有的数据排除不正确的假设

仔细分析已有的数据，寻找矛盾，力求排除前一步列出的假设。如果所有列出的假设都被排除了，则需要提出新的假设，如果余下的假设多于一个，则首先选取最有可能成为出错原因的那一个假设。

### (3) 精化余下的假设

利用已知的线索进一步精化余下的假设，使之更具体化，以便精确确定故障的位置。

### (4) 证明余下的假设

这一步非常重要，具体做法与归纳法的第4步相同。

## 9.6 自动测试工具

软件测试是一件既繁重又困难，并且质量要求又高的脑力劳动，为了减轻测试工作的劳动强度，提高测试工作效率，确保测试工作质量，人们开发出了许多的自动测试工具。例如测试数据生成程序、静态分析程序、动态分析程序、模块测试程序等。

### 1. 测试数据生成程序

测试数据生成程序主要用于为某项测试自动产生输入数据。

可以使用测试数据生成程序随机产生测试用例，由此可防止因为主观倾向对测试用例设计的影响。

当需要测试某个系统在实际环境中的工作性能时，测试数据生成程序特别有用。例如，为了测试某个数据库应用系统的数据查询性能，往往需要为该项测试输入大量的数据，这些数据就可以使用测试数据生成程序快速产生。

### 2. 动态分析程序

动态分析程序的主要功能是分析被测程序中每个语句的执行次数，可以用于软件测试与调试。例如，可以通过动态分析程序发现测试过程中没有执行的语句，以便增加相应的测试数据。在调试过程则可以利用它发现那些不按预定要求终结的循环，发现不应该执行而实际上执行了的代码，或者应该执行而实际上没有执行的代码。

动态分析程序一般包含两个基本部分：

(1) 检测部分：用于向被分析程序中插入检测语句，当程序执行时，这些语句收集和整理有关每个语句执行次数的信息。

(2) 显示部分：用于汇集检测语句提供的信息，并以某种容易理解的形式输出这些信息。

### 3. 静态分析程序

静态分析程序不需要执行被测试的程序，它仅仅扫描被测试程序的正文，从中寻找可能导致错误的异常情况。例如，使用了一个尚未赋值的变量；赋了值的变量始终未被使用；实际参数与形式参数的类型或个数不符；永远执行不到的程序段；等等。



#### 4. 模块测试程序

模块测试程序可以提供一个单元模块测试平台。

在测试模块时，必须考虑测试模块与其他模块之间存在调用与被调用关系，因此，需要为它编写驱动模块和桩模块，以代替它的上下层有关模块。渐增组装测试可以减少这类模块的编制任务，模块测试程序就是一个承担生成这些辅助性模块的工具。它提供了一种专门的测试用例描述语言，负责将输入数据传送给所测试模块，然后将实际输出结果与在描述测试用例的语言中所表述的期望结果进行比较，由此可以找出程序错误。

#### 5. 集成测试环境

用于测试的自动工具很多，除上面介绍的工具以外，还有环境模拟程序、代码检查程序、测试文档生成程序、测试执行验证程序、输出比较程序、程序正确性证明程序以及各种调试工具。集成测试环境的作用就是将诸多测试工具集成到一个统一的测试环境中，由此使得各种不同的测试工具可以协作工作，并能够有效提高测试质量与测试效率。

## 9.7 软件可靠性评估

软件测试的根本目标是消除故障保证软件的可靠性。因此，在软件测试阶段，往往需要根据有关测试数据的统计分析结果，对软件的可靠性进行一定的评估。

### 9.7.1 可靠性概念

#### 1. 软件可靠性的定义

软件可靠性的一般性定义是：程序在给定的时间间隔内，按照规格说明书的规定成功地运行的概率。

在上述定义中包含的随机变量是时间间隔。显然，随着运行时间的增加，运行时遇到程序故障的概率也将增加，即可靠性随着给定的时间间隔的加大而减少。

#### 2. 软件的可用性

软件的可用性是指软件系统可以使用的程度。一般说来，对于任何其故障可以修复的系统，都应该同时使用可靠性和可用性衡量它的优劣程度。

软件可用性的一般性定义是：程序在给定的时间点，按照规格说明书的规定，成功地运行的概率。

可靠性和可用性之间的主要差别是，可靠性意味着在 0 到  $t$  这段时间间隔内系统没有失效，而可用性只意味着在时刻  $t$ ，系统是正常运行的。

为了方便可用性的计算，一般使用稳态可用性对系统进行可用性评价。

系统稳态可用性计算式是：

$$A_{ss} = MTTF / (MTTF + MTTR)$$

其中， $MTTF$  是系统平均无故障时间， $MTTR$  是系统平均维修时间。

### 9.7.2 估算系统平均无故障时间

软件的平均无故障时间  $MTTF$  是一个重要的质量指标，作为对软件的一项要求， $MTTF$  往往

由用户提出，并需要进行估算。

在估算  $MTTF$  的过程中需要使用下述符号表示有关的数量。

$E_T$  —— 测试之前程序中的故障总数。

$I_T$  —— 程序长度（机器指令总数）。

$t$  —— 测试（包括调试）时间。

$E_d(t)$  —— 在 0 至  $t$  期间发现的错误数。

$E_c(t)$  —— 在 0 至  $t$  期间改正的错误数。

估算平均无故障时间的计算式是：

$$MTTF = 1 / (K (E_T / I_T - E_c(t) / I_T))$$

其中，计算式中的  $K$  为经验常量，它的典型值是 200。

### 9.7.3 估算系统中的故障总数

程序中潜藏的故障的数目  $E_T$  是一个十分重要的量，它既影响软件的可靠程度，又是计算软件平均无故障时间的重要参数。

比较常用的  $E_T$  估计方法是植入故障法。

为了使用植入故障法进行  $E_T$  估算，在测试之前需要由专人在程序中随机地植入一些故障，测试之后，根据测试小组发现的故障中原有的和植入的两种故障的比例，来估计程序中原有故障的总数  $E_T$ 。

假设人为地植入的故障数为  $M$ ，经过一段时间的测试之后发现了  $m$  个植入的故障和  $n$  个原有的故障。如果可以认为测试方案发现植入故障和发现原有故障的能力相同，则能够估计出程序中原有故障的总数是：

$$N = M \times n / m$$

其中的  $N$  即是系统故障总数  $E_T$  的估算值。

## 小 结

### 1. 测试目标

尽力发现软件中的错误，而不是为了验证软件的正确性。

### 2. 测试方法

(1) 黑盒测试：基于程序的外部功能规格而进行的测试，又称为功能测试。

(2) 白盒测试：基于程序的内部结构与处理过程而进行的测试，又称为结构测试。

### 3. 单元测试

单元测试的对象是单元模块，一般以白盒测试为主，以黑盒测试为辅。测试内容包括模块接口测试、局部数据结构测试、路径测试、错误处理测试、边界测试。

单元测试通常在编码阶段进行。测试时需要用到辅助模块，如驱动模块、桩模块。

### 4. 集成测试

系统集成时主要有非渐增组装测试和渐增组装测试这两种方法：

(1) 非渐增组装测试：一种一次性地进行系统组装的方法。

(2) 渐增组装测试：一种将单元模块的确认测试与集成测试结合在一起的测试方法，它比非渐增组装测试是具有更大的优越性。可以自顶向下渐增集成，也可以自底向上渐增集成。

#### 5. 确认测试

确认测试又称有效性测试，其任务是验证软件的功能、性能及其他特性是否与用户的要求一致。在进行确认测试时，可以采用 Alpha 测试或 Beta 测试。其中，Alpha 测试是在开发环境下由用户进行的测试，而 Beta 测试则是由软件用户在软件实际使用环境下进行的测试。

#### 6. 测试用例设计

设计测试用例就是为测试准备测试数据。由于测试用例不同，发现程序错误的能力也就不同，为了提高测试效率降低测试成本，应该选用高效的测试用例。

白盒测试用例设计主要采用逻辑覆盖，包括语句覆盖、判定覆盖、条件覆盖、判定—条件覆盖、条件组合覆盖和路径覆盖。

黑盒测试用例设计包括等价划分、边界值分析和错误推测等几种方法。

#### 7. 面向对象测试

##### (1) 面向对象单元测试

不能孤立地测试单个操作，而应该把操作作为类的一部分来测试。

##### (2) 面向对象集成测试

- 基于线程的测试。
- 基于使用的测试。

##### (3) 面向对象确认测试

研究系统的用例模型和活动模型，设计出确认测试时的用户操作脚本。

#### 8. 软件调试

软件调试也叫做排错，涉及诊断与排错这两个步骤。但调试的关键是诊断。

常用的调试方法有：输出存储器内容、在程序中插入输出语句、使用自动调式工具。

常用的调试策略有：试探法、回溯法、对分查找法、归纳法、演绎法。

#### 9. 自动测试工具

常用的自动测试工具有：测试数据生成程序、动态分析程序、静态分析程序、模块测试程序。

#### 10. 软件可靠性评估

软件可靠性的定义是：程序在给定的时间间隔内，按照规格说明书的规定成功地运行的概率。

软件可用性的定义是：程序在给定的时间点，按照规格说明书的规定，成功地运行的概率。为了方便可用性的计算，一般使用稳态可用性对系统进行可用性评价。

系统平均无故障时间的估算式是： $MTTF = 1 / (K (E_T / I_T - E_c(t) / I_T))$

## 习 题

1. 简述你对软件测试目标的认识。
2. 什么是黑盒测试？什么是白盒测试？

3. 说明软件配置在软件测试中的用途。
4. 描述单元测试的对象、方法与内容。
5. 什么是非渐增组装测试？什么是渐增组装测试？
6. 试比较自顶向下渐增与自底向上渐增的异同？
7. 什么是确认测试？该测试过程需要经过哪几个步骤？
8. 试对 Alpha 测试与 Beta 测试进行比较？
9. 什么是逻辑覆盖？主要有哪几种逻辑覆盖？
10. 试对等价类划分与边界值分析这两种黑盒测试方法进行比较？
11. 试描述错误推测方法的特点及其作用？
12. 试比较面向对象单元测试与传统结构化单元测试的异同？
13. 回溯法程序调试策略有什么特点？
14. 归纳法程序调试策略有什么特点？
15. 测试过程中如何对软件可靠性进行评估？
16. 什么是植入故障法？它有什么用途？
17. 试分别使用语句覆盖、条件覆盖设计学生等级成绩输出程序的测试用例。

该程序伪码如下：

```
PROCEDURE Grade ( INTEGER x )
```

```
    IF x > 100 OR x < 0
```

```
    THEN
```

```
        PRINT( “ 传入数据无效 ” )
```

```
    END IF
```

```
    IF x >= 85 AND x <= 100
```

```
    THEN
```

```
        PRINT( “ 优秀 ” )
```

```
    END IF
```

```
    IF x >= 60 AND x < 85
```

```
    THEN
```

```
        PRINT( “ 合格 ” )
```

```
    END IF
```

```
    IF x >= 0 AND x < 60
```

```
    THEN
```

```
        PRINT( “ 不合格 ” )
```

```
    END IF
```

```
END Grade
```

18. 需要对一个用户注册窗口进行黑盒测试。假如用户注册码规定为 4~8 位字符，试分别使用等价类划分、边界值分析这两种方法设计该软件问题的测试用例。

## 第10章 软件维护

在软件系统开发完成并交付用户使用以后，软件就进入到了运行维护阶段。

尽管运行维护时期的软件已经脱离了开发环境，但是与软件开发有关的一系列工作则仍在继续。例如，软件运行过程中出现了错误，因此需要检查错误并改造错误；也可能是用户有了新的需求想法，需要为软件增加一些功能模块，或者将软件系统转移到一个新的运行环境下工作等。这一系列的活动就是软件维护。

通常情况下，诸多软件维护工作仍由原来的软件开发机构负责，但也有可能是用户自己承担维护任务。然而，无论是谁在最终承担着软件维护的具体任务，对软件维护问题的考虑则是从软件问题被提出开始就已经涉及了，并贯穿于整个软件生命周期之中，因为只有这样，才能使软件真正具有可维护的特性，并使软件维护变得便利起来。

### 10.1 软件维护概述

#### 10.1.1 软件维护定义

一般认为，软件维护就是在软件运行维护阶段，为了改正软件错误，或为了满足用户新的应用需要，而对软件进行改错、变更或进化的过程。

具体地说，软件维护涉及以下几个方面的任务。

(1) 改正性维护：由于软件测试技术的限制，已投入使用的软件必然会有一些隐藏的错误或缺陷。这些隐藏下来的错误或缺陷，在某些特定的使用环境下可能会暴露出来，并有可能影响到软件的正常使用。因此，软件技术人员需要对暴露出来的软件错误进行诊断，并设法改正这个错误。这个诊断与改正错误的过程，就叫做改正性维护。

(2) 适应性维护：随着计算机技术的飞速发展，软件的工作环境，例如硬件设备、软件配置、数据环境、网络环境等，都有可能发生变化。为了使软件适应这种变化，往往需要对软件进行改造。这个为使软件适应新的工作环境而对软件进行改造的过程，就叫做适应性维护。

(3) 完善性维护：在软件使用过程中，用户难免会对软件提出一些新的与完善软件有关的要求，例如，要求增加一些新的功能，要求对系统原有的功能关系做一些调整，要求提高数据检索速度，要求操作界面更加人性化等。而为了满足这些要求，就必须对软件进行改造，以使软件在功能、性能、界面等方面有所进化。由于这些原因而对软件进行的维护活动，就叫做完善性维护。

大多数软件维护活动的表现是：在软件运行阶段初期，改正性维护的工作量较大，而随着软件错误发现率的降低，软件系统的工作逐步趋于稳定，改正性维护也就由此下降。然而，随着软件使用时间的增加，用户新的需求意愿会逐渐形成并提出，于是软件适应性维护和完善性维护的工作量就会逐步增加起来。

除了上述三种类型的维护活动之外，还有一种叫做预防性维护的活动。这是为了使软件具有更好的可维护性、可靠性，或为了今后软件进化的便利，而进行的一系列与维护有关的准备性工作。

有关统计数据表明，在上述几种维护活动中，完善性维护所占的比重最大，约占整个维护工作的 50% 以上。也就是说，大部分的软件维护工作是扩充功能、提高性能，而不是改正错误。预防性维护则只占很小的比例。

### 10.1.2 影响软件维护工作的因素

有关统计数据显示，软件维护活动所消耗的工作量占整个软件生存期工作量的 70% 以上。许多软件开发机构就因为软件维护工作量的巨大，而导致新的软件项目不能承接，新的软件产品不能及时开发。

软件维护需要消耗这么大的工作量，其原因是什么呢？

有关研究表明，影响软件维护工作量的原因，归纳起来主要有以下几个方面。

(1) 系统大小：软件系统越大，其执行功能越复杂，理解掌握起来越困难。因而需要更多的维护工作量。

(2) 程序设计语言：许多软件是用较老的程序设计语言编写的，程序逻辑复杂、混乱，而且没有做到模块化和结构化，直接影响到程序的可读性与可维护性。

(3) 系统文档：一些系统在开发时并没有考虑到将来维护的便利，并没有按照软件工程的要求进行开发，因而没有文档，或文档太少，或在长期的维护过程中文档在许多地方与程序实现变得不一致，这样在维护时就会遇到很大困难。

(4) 系统年龄：老系统比新系统需要更多的维护工作量。随着不断的修改，老系统结构变得越来越乱；由于系统维护人员经常更换，程序变得越来越难于理解。

(5) 其他因素：包括应用的类型、数学模型、任务的难度、开关与标记、IF 嵌套深度、索引或下标数等，它们都会给维护工作带来影响。

### 10.1.3 非结构化维护与结构化维护

#### 1. 非结构化维护

非结构化维护往往与早期软件非工程化开发有关系，是软件开发过程中没有按照软件工程原则实施软件开发的后遗症。

许多早期软件，由于没有按照软件工程原则实施软件开发，以致和软件配套的一系列文档没有建立起来，保留下来的可能只有源程序。

应该说，软件开发过程中文档的完整性，对软件今后的维护有非常大的影响。如果软件配置仅仅只有源程序代码，那么软件维护活动就需要直接从源程序代码开始。显然，面对这样的软件进行维护，将会是困难重重，而且往往还会使程序变得更加混乱，更加不能理解。

#### 2. 结构化维护

软件工程所要求的是结构化维护，它建立在严格按照软件工程原则实施软件开发基础上，因此各个阶段的文档完整，能够比较全面地说明软件的功能、性能、软件结构、数据结构、系统接口和设计约束等，这些都将给今后软件的维护带来便利。

实际上,结构化维护就是一种依靠完整的软件配置而进行的维护,其中的软件配置包括需求规格说明、设计说明、测试说明、源程序清单和维护计划等诸多文档,因此,结构化维护可以从评价文档开始。例如,通过对设计说明的评价,由此确定软件重要的结构特点、性能特点以及接口特点;估量所要求的改动将给软件带来的影响,并为维护实施途径制定出合适的计划。

而在软件维护具体实施过程中,则可以先修改设计,并且对所做的改动进行仔细复查,接下来编写相应的源程序代码,然后再依据测试说明书中包含的信息进行回归测试,最后把修改后的软件再次交付使用。

很显然,结构化的维护是一种有利于系统健康发展的维护,并能够在减少维护工作量、提高维护效率等方面产生积极作用。

#### 10.1.4 软件维护的代价

软件维护代价包括有形与无形这两个方面的代价。其中,有形代价是指软件维护的直接费用支出,无形代价则指其他非直接的维护代价。

例如以下方面的无形代价:

- (1) 一些看起来合理的修复或修改请求由于不能及时安排,以致用户不满意。
- (2) 维护带来了软件变更,一些新的错误由此被引入软件,使得软件整体质量下降。
- (3) 软件开发人员经常被抽调到维护工作中去,由此使得软件开发工作受到干扰。

软件的维护代价还反映到了软件生产率上,它使得软件生产率显著下降。这种情况在维护旧系统时常常遇到。如据 Gausler 在 1976 年的报道,美国空军的飞行控制软件每条指令的开发成本是 75 美元,然而维护成本大约是每条指令 4000 美元,也就是说,生产率下降了 50 倍以上。

软件维护的有形代价,即维护直接支出,则可以按照下面的模型进行估算。

可以将用于维护工作的劳动可以分成生产性活动(例如分析评价,修改设计和编写程序代码等)和非生产性活动(例如理解程序代码的功能,解释数据结构、接口特点和性能限度等)。依照这种分类方法,可以使用下面的表达式计算维护工作量:

$$M = P + K \times \exp(C - D)$$

其中, $M$ 是维护用的总工作量, $P$ 是生产性工作,量, $K$ 是经验常数, $C$ 是软件系统复杂程度(非结构化设计和缺少文档都会增加软件的复杂程度), $D$ 是维护人员对软件的熟悉程度。

上面的模型表明,如果软件的开发途径不好,例如,没有按照软件工程方法进行软件开发,而且原来的开发人员不能参加维护工作,那么软件维护工作量将指数地增加。

## 10.2 软件可维护性

软件可维护性是指维护人员理解、改正、改动和改进这个软件的难易程度。

对于一个复杂的软件系统,造成其维护工作困难的一个直接原因是缺乏软件开发文档。由于没有足够的、规范的文档,因此很难理解以前的软件的功能、算法,源程序很难阅读和理解,以致软件的可维护性很差。因此,为了使软件能够易于维护,首先必须考虑使软件具有很好的可维护性。

应该说,软件的可维护性是衡量软件质量的重要指标,是软件开发阶段各个时期都应该重

视的关键目标。因此有必要对软件的可维护性进行有效评估。

一个应用广泛的可维护性评估模型是：通过对可理解性、可靠性、可测试性、可修改性、可移植性、运行效率和可使用性这七个方面的软件特性的评价，而对软件的可维护性进行综合评估。

下面是对这七个方面特性的说明：

(1) 可理解性：指人们通过阅读源代码和相关文档，了解程序功能及其如何运行的难易程度。一个可理解的程序应该具有模块化、风格一致、结构完整等特性。

(2) 可靠性：指程序按照用户的要求和设计目标，在给定的一段时间内正确执行的概率。其度量标准有：平均失效间隔时间 ( $MTTF$ )、平均修复时间 ( $MTTR$ )。

(3) 可测试性：指诊断程序错误的难易程度。对于程序模块，可用程序复杂性来度量可测试性。程序的环路复杂性越大，程序的路径就越多，全面测试程序的难度就越大。

(4) 可修改性：指程序修改的难易程度。一个可修改的程序应当是可理解的、通用的、灵活的、简单的。

(5) 可移植性：指程序转移到一个新的计算环境的可能性的的大小。一个可移植的程序应具有结构良好、灵活，并具有与计算机、操作系统无关的特点。

(6) 运行效率：指一个程序能执行预定功能而又不浪费机器资源的程度。这些机器资源包括：内存容量、外存容量、通道容量和执行时间。

(7) 可使用性：指对于用户而言，程序的方便、实用和易于使用的程度。

需要注意的是，上述七个方面的软件特性，对于不同类型的软件维护，会有不同的侧重表现。表 10-1 显示了各类维护中应该侧重的特性。

表 10-1 各类维护需要侧重的特性

特 性	改正性维护	适应性维护	完善性维护
可理解性	Y		
可测试性	Y		
可修改性	Y	Y	
可靠性	Y		
可移植性		Y	
可使用性		Y	Y
效率			Y

## 10.3 软件维护的实施

### 10.3.1 维护机构

随着软件维护工作量的不断增加，许多软件开发单位开始意识到了设立软件维护机构的重要性。这种维护机构有可能是一个临时维护小组，也有可能是一个长期专门从事软件维护的职能部门。

一个临时维护小组往往被派去执行一些特殊的或临时的维护任务。例如，当正在工作的软



件系统出现了不能回避的严重运行错误时，可能需要临时组织一个维护小组前往用户单位对系统进行排错检查。

对于一个需要长期稳定运行的复杂系统，维护工作需要有一个相对稳定的维护部门来完成。一般说来，执行长期维护职能的维护部门在系统开发完成之前就应该成立，并需要有严格的组织与管理规则，以确保今后维护工作的顺利开展。

一项维护工作，无论是临时的还是长期的，都往往会涉及以下人员或角色：

(1) 维护机构负责人：全权负责所有维护活动，包括技术与管理两个方面的工作，并负责向上级主管部门报告维护工作的开展情况。

(2) 系统监督员：负责对维护申请进行技术性评价，以确保维护的有效性。

(3) 配置管理员：进行与软件维护有关的软件配置管理。

(4) 维护管理员：负责同软件开发部门或其他部门的联系，收集、整理有关维护的信息。

(5) 维护技术人员：负责分析程序错误、进行程序修正。

为使维护工作正常开展，上述维护人员需要协作工作，例如可以按照下面的协作关系与工作步骤实施对软件的维护。

(1) 有关人员将维护申请报告表提交给维护管理员登记。

(2) 维护管理员把维护申请报告交系统监督员进行技术性评价。

(3) 系统监督员从技术角度对该项维护的可行性、必要性等做出说明。

(4) 在得到系统监督员的技术性评价之后，维护管理员把维护申请报告表提交给维护机构负责人。

(5) 维护机构负责人将根据对维护申请报告的技术评价，决定如何进行软件维护。

(6) 维护机构负责人需要将维护决定通知维护管理员，以便维护管理员能够及时安排相关技术人员实施维护。

(7) 维护机构负责人还需要将维护决定通知配置管理员，以便技术人员在对系统进行维护的过程中，配置管理员能够严格把关，控制维护范围，并对软件配置进行审计。

图 10-1 是维护工作人员之间的协作关系图示说明。

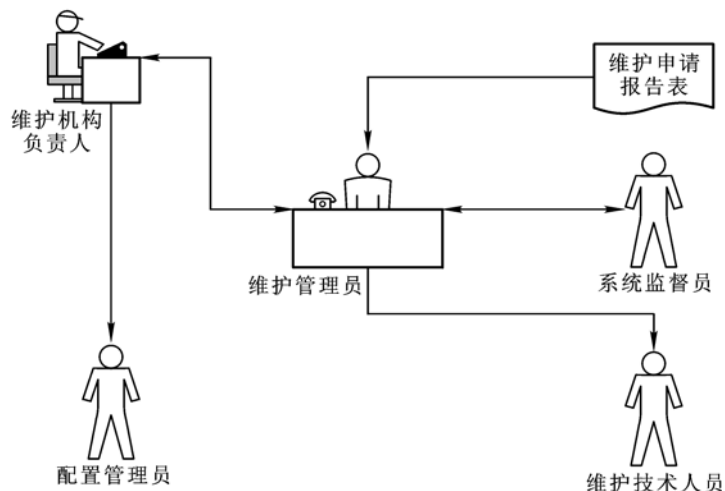


图 10-1 维护工作人员之间的协作关系

### 10.3.2 维护申请报告

为使维护按规程进行，维护需要先以文档的形式提出申请，例如，由申请维护的人员（用户、开发人员）填写一份软件维护申请报告表。

对于改正性维护，申请报告必须尽量完整地说明错误产生的情况，包括运行时的环境、输入数据、错误提示等。

对于适应性或完善性的维护，则应该提交一份简要的维护要求说明。

一切维护活动都应该从维护申请报告开始。并需要由维护机构对维护请求进行评审。由此确定维护类型（改正性维护、适应性维护或完善性维护），然后根据需要维护的软件问题的严重性，对维护作出具体的工作安排。

在维护过程中，软件维护机构内部还应该制定一份软件修改报告，该报告是维护阶段的技术性文档，其一般包含以下信息：

- （1）维护工作量。
- （2）维护类型。
- （3）维护的优先顺序。
- （4）预见的维护结果。

### 10.3.3 软件维护工作流程

软件维护的工作流程如图 10-2 所示。其主要工作步骤如下：

（1）首先需要确定的是确定维护类型。由于用户的看法可能会与维护人员的评价不一致，当出现意见不一致时，维护人员应该与用户进行协商。

（2）对于改正性维护申请，需要先对错误的严重性进行评价。如果存在严重的错误，则必须立即安排维护人员进行“救火”式的紧急维护。而对于不太严重的错误，则可根据任务情况和问题的严重程度列入维护计划，按优先顺序统一安排维护时间。

（3）对于适应性维护和完善性维护申请，需要先确定每项申请的优先次序。若某项申请的优先级非常高，就可立即开始维护工作，否则，将维护申请纳入软件开发任务计划进行排队（适应性维护与完善性维护可当作开发看待），统一安排维护时间。

尽管维护申请的类型不同，但都要进行同样的技术工作。这些工作有：修改软件需求说明、修改软件设计、设计评审、对源程序做必要的修改、单元测试、集成测试（回归测试）、确认测试、软件配置评审等。

在每次软件维护任务完成之后，应该对维护情况进行评审。

评审内容包括：

- （1）设计、编码、测试中的哪些方面还可以改进？
- （2）哪些维护资源应该有，但事实上却没有？
- （3）维护工作中主要的或次要的障碍是什么？
- （4）是否需要考虑预防性维护？

维护情况评审对今后维护工作的进行有重要的影响，并可为软件机构的有效管理提供重要的反馈信息。

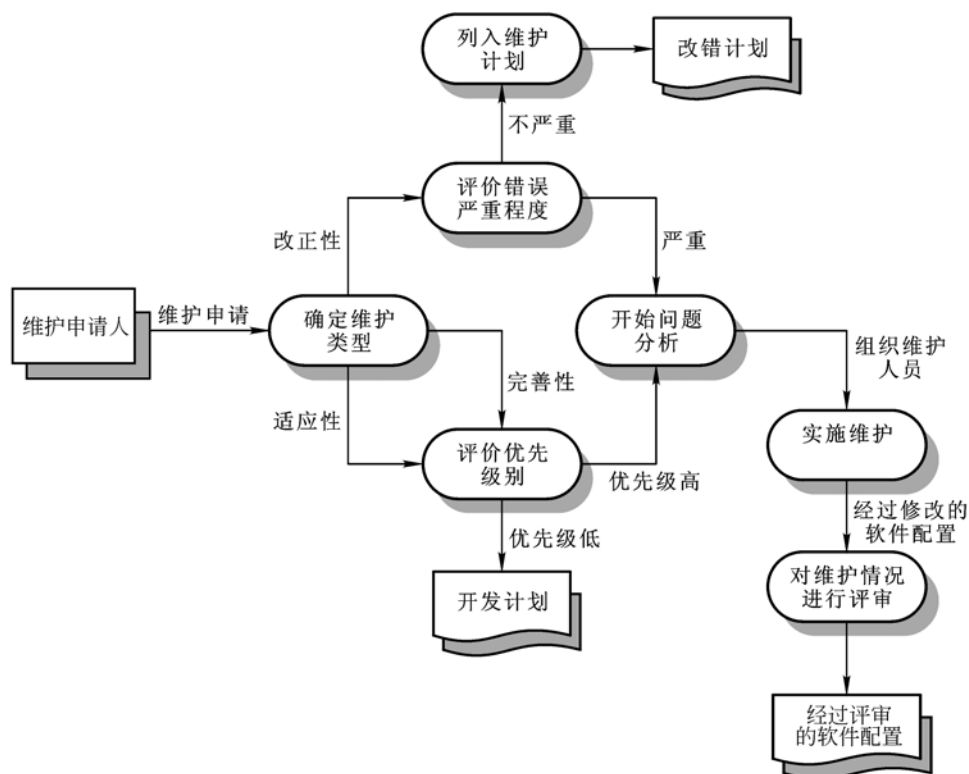


图 10-2 软件维护工作流程

### 10.3.4 维护记录

为了估计软件维护的有效程度，确定软件产品的质量，同时确定维护的实际开销，需要在维护的过程中做好维护档案记录。

维护记录内容包括：程序名称、源程序语句条数、机器代码指令条数、所用的程序设计语言、程序安装的日期、程序安装后的运行次数、与程序安装后运行次数有关的处理故障次数、程序改变的层次及名称、修改程序所增加的源程序语句条数、修改程序所减少的源程序语句条数、每次修改所付出的“人时”数、修改程序的日期、软件维护人员的姓名、维护申请报告的名称、维护类型、维护开始时间和维护结束时间、花费在维护上的累计“人时”数、维护工作的净收益等。

### 10.3.5 维护评价

由于缺乏可靠的数据，评价维护活动往往比较困难。但如果维护的档案记录做得比较好，就可以得出一些维护“性能”方面的度量值。

可参考的度量值如：

- (1) 每次程序运行时的平均出错次数；
- (2) 花费在每类维护上的总“人时”数；
- (3) 每个程序、每种语言、每种维护类型的程序平均修改次数；

- (4) 因为维护, 增加或删除每个源程序语句所花费的平均“人时”数;
- (5) 用于每种语言的平均“人时”数;
- (6) 维护申请报告的平均处理时间;
- (7) 各类维护申请的百分比。

这七种度量值提供了定量的数据, 据此可对开发技术、语言选择、维护工作计划、资源分配以及其他许多方面做出判定。因此, 这些数据可以用来评价维护工作。

## 10.4 对老化系统的维护

老化系统是指一些使用早期程序设计语言(如 FORTRAN、COBOL)开发的系统。由于使用时间长, 以前开发这些系统的技术人员可能已经离开了开发机构, 并且这些系统大都没有使用现代先进的开发方法与技术, 因此, 系统的结构很差, 文档也不完整, 都曾经历过维护变更, 并且大都没有保存维护时的修改记录。

对于这样的没有完整软件配置和良好设计的老化系统, 维护已经变得非常困难, 但是由于某些方面的原因, 目前还不能将它们完全抛弃, 因此, 也就必须对它们做一些必要的维护。为了能够有效地维护它们, Yourdon 提出了以下的几点维护建议:

- (1) 在进入“紧急维修”之前, 必须研究程序的使用环境及有关资料, 尽可能得到更多的背景信息。
- (2) 力图熟悉程序的所有控制流程。最初可以忽略某些编码细节。如果设计存在, 则可利用它们来帮助画出结构图和高层流程图。
- (3) 评价现有文档的可用性。若有帮助, 可利用它们在源程序清单中插入注释。
- (4) 充分利用交叉引用表、符号表及其他由编译程序或汇编程序等提供的交叉引用信息。
- (5) 必须非常谨慎地对程序进行修改。如有可能, 应该尊重程序原有的风格和格式, 要说明需要变更的程序指令。
- (6) 在删除某些代码时, 要确认代码确实不再使用。
- (7) 不要试图共享程序已有的临时变量或工作区, 需要时可插入新的变量以避免冲突。
- (8) 保持详细的维护活动和维护结果记录。
- (9) 如果程序结构混乱, 修改受到干扰, 可抛弃程序重新编写。
- (10) 插入出错检验。

## 10.5 逆向工程与再工程

通常意义上的软件开发是正向工程, 其过程方向是从设计到产品。然而, 逆向工程的过程方向则与此正好相反, 它是从源程序, 甚至是从目标程序, 到设计模型。

可以把逆向工程描述为一个魔术管道, 从管道一端流入的是一些非结构化的无文档的源代码或目标代码, 而从管道另一端流出的则是计算机软件的分析、设计文档。

理想情况情况下, 逆向工程过程应当能够从源程序中反向导出程序流程设计(最低层抽象)、数据结构(低层抽象)、数据和控制流模型(中层抽象)和实体联系模型(高层抽象)。因

此，逆向工程过程可以给软件工程师带来许多有价值的信息，例如，可以使对程序的理解变得更加容易。

但在实际应用中，逆向工程的反向导出能力一般都会随着抽象层次的增加而减少。例如，给出一个源程序清单，也许可以利用它得到比较完整的程序流程设计；但是，要通过它得到一个比较完整的数据流图，则就比较困难了。

逆向工程的最初用途是用来破解竞争对手的软件产品的秘密。但在目前，逆向工程已被更多地用到了工程创建上，如 Rational Rose、Microsoft Visio，它们作为 UML 建模工具，就都具有逆向工程的创建功能。其作用是能够从已经存在的软件系统中，反向导出它们的设计模型，以方便 UML 建模方法所需要的从分析到设计，再到实现的多次迭代。

图 10-3 所示是逆向工程的工作流程图。

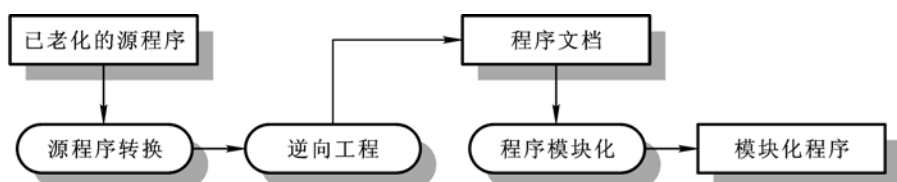


图 10-3 逆向工程

逆向工程也被应用到了软件维护上，例如那些老化系统，由于只有源程序，缺少设计说明，因此系统维护非常困难。然而，逆向工程则有可能从老化系统的源代码中提取程序流程设计、系统结构设计，甚至是数据流图，由此能够给老化系统的维护带来方便。

当逆向工程被用于重新构造或重新生成老化系统时，这个过程就叫做再工程。再工程不仅能从已存在的程序中重新获得设计信息，而且还能使用这些信息来改建或重建现有的系统，例如给系统增加一些新的功能或改善系统的性能，使系统的综合质量有所提高。

图 10-4 所示是再工程的工作流程图。

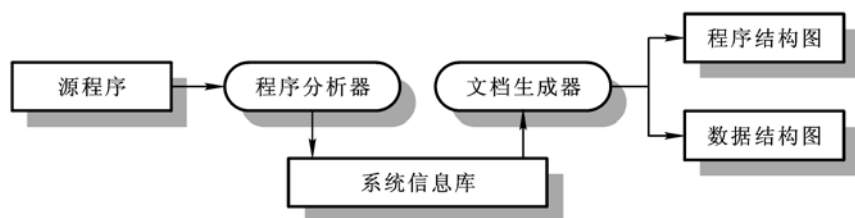


图 10-4 再工程的工作流程

## 10.6 软件配置管理

软件配置管理是一组针对软件产品的追踪和控制活动，它贯穿于软件生命周期的始终，并代表着软件产品接受各项评审。

当对软件进行维护时，软件产品发生了变化，这一系列的改变，必须在软件配置中体现出来，以防止因为维护所产生的变更给软件带来混乱。

软件开发过程中，需要输出的信息有以下三种：(1) 计算机程序；(2) 描述计算机程序的文档；(3) 数据结构。软件配置就由这些信息所组成。

### 10.6.1 配置标识

为了方便对软件配置中的各个对象进行控制与管理，首先应给它们命名，再利用面向对象的方法组织它们。通常需要标识两种类型的对象：基本对象和复合对象。基本对象是由软件工程师在分析、设计、编码和测试时所建立的“文本单元”。复合对象则是基本对象或其他复合对象的一个收集。

每个对象可用一组信息来惟一地标识它，这组信息包括名字、描述、资源、实现等内容。

### 10.6.2 变更控制

软件生命期内全部的软件配置是软件产品的真正代表，必须使其保持精确。软件工程过程中某一阶段的变更，均要引起软件配置的变更，这种变更必须严格加以控制和管理，以保证修改信息能够精确、清晰地传递到软件工程过程的下一步骤。

变更控制包括建立控制点和建立报告与审查制度。

在此过程中，首先用户提交书面的变更请求，详细申明变更的理由、变更方案、变更的影响范围等。然后由变更控制机构确定控制变更的机制、评价其技术价值、潜在的副作用、对其他配置对象和系统功能的综合影响以及项目的开销，并把评价的结果以变更报告的形式提交给变更控制负责人进行变更确认。

软件的变更通常有两类不同的情况：

(1) 为改正小错误需要的变更。它是必须进行的，通常不需要从管理角度对这类变更进行审查和批准。但是，如果发现错误的阶段在造成错误的阶段后面，例如在实现阶段发现了设计错误，则必须遵照标准的变更控制过程，把这个变更正式记入文档，把所有受这个变更影响的文档都做相应的修改。

(2) 为了增加或者删掉某些功能、或者为了改变完成某个功能的方法而需要的变更。这类变更必须经过某种正式的变更评价过程，以估计变更需要的成本和它对软件系统其他部分的影响。如果变更的代价比较小且对软件系统其他部分没有影响或影响很小，通常应批准这个变更。反之，如果变更的代价比较高或者影响比较大，则必须权衡利弊，以决定是否进行这种变更。如果同意这种变更，需要进一步确定由谁来支付变更所需要的费用。如果是用户要求的变更，则用户应支付这笔费用。否则，必须完成某种成本/效益分析，以确定是否值得做这种变更。应该把所做的变更正式记入文档，并相应地修改所有相关的文档。

### 10.6.3 版本控制

软件变更往往会带来软件版本的改变与新版本的发布，对此，需要进行有效的控制。

版本控制往往利用工具来进行管理与标识，并有许多不同的版本控制自动方法。图 10-5 所示是用于表现系统不同版本的演变图。图中的各个结点都是一个用于反映版本完整组成的聚合对象，是源代码、文档、数据的一次完整收集。

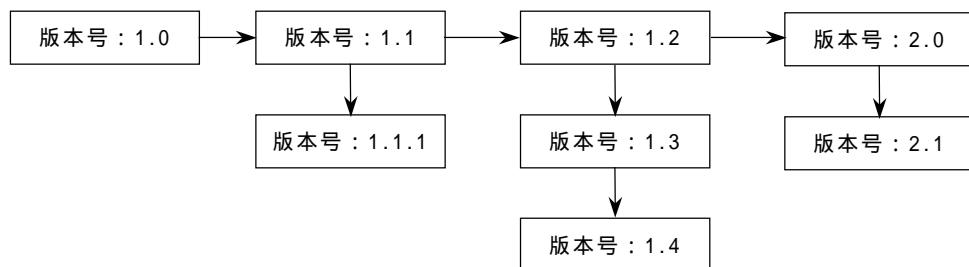


图 10-5 版本演变

## 小 结

### 1. 软件维护定义

软件维护是在软件运行维护阶段，为了改正软件错误或为了满足用户新的应用需要，而对软件进行改错、变更或进化的过程。

维护任务一般分为：改正性维护、适应性维护、完善性维护和预防性维护。

### 2. 影响软件维护工作的因素

主要因素有：系统大小、程序设计语言、系统文档和系统年龄等。

### 3. 非结构化维护

没有按照软件工程原则实施软件开发，以致和软件配套的一系列文档没有建立起来，保留下来的可能只有源程序。

### 4. 结构化维护

建立在严格按照软件工程原则实施软件开发基础上，因此各个阶段的文档完整，能够比较全面地说明软件的功能、性能、软件结构、数据结构、系统接口和设计约束等。

### 5. 软件维护的代价

软件维护代价包括有形与无形这两个方面的代价。其中，有形代价是指软件维护的直接费用支出，无形代价则指其他非直接的维护代价。

### 6. 软件可维护性

软件可维护性是指维护人员理解、改正、改动和改进这个软件的难易程度。

可以从系统的可理解性、可靠性、可测试性、可修改性、可移植性、运行效率和可使用性这七个方面对软件的可维护性进行综合评估。

### 7. 软件维护的实施

软件维护实施过程中，一般涉及以下几个问题：维护机构、维护申请报告、软件维护工作流程、维护记录和维护评价。

### 8. 对老化系统的维护

老化系统是指一些使用早期程序设计语言开发的系统。为了能够有效地对老化系统进行维护，Yourdon 提出了以下的几点维护建议：

(1) 尽可能得到更多的背景信息。

- (2) 力图熟悉程序的所有控制流程。
- (3) 评价现有文档的可用性。
- (4) 充分利用交叉引用信息。
- (5) 必须非常谨慎地对程序进行修改。
- (6) 在删除某些代码时,要确认代码确实不再使用。
- (7) 不要试图共享程序已有的临时变量或工作区。
- (8) 保持详细的维护活动和维护结果记录。
- (9) 如果程序结构混乱,修改受到干扰,可抛弃程序重新编写。
- (10) 插入出错检验。

#### 9. 逆向工程与再工程

逆向工程是通过源程序,甚至是目标程序,由此导出设计模型、分析模型的过程。可以把逆向工程描述为一个魔术管道,从管道一端流入的是一些非结构化的无文档的源代码或目标代码,而从管道另一端流出的则是计算机软件的分析、设计文档。

逆向工程被用到了软件维护上,通过从老化系统的源代码中提取程序流程设计、系统结构设计,甚至是数据流图,给老化系统的维护带来方便。

当逆向工程被用于重新构造或重新生成老化系统时,这个过程就叫做再工程。再工程不仅能从已存在的程序中重新获得设计信息,而且还能使用这些信息来改建或重建现有的系统。

#### 10. 软件配置管理

配置管理包括软件配置标识、软件变更控制和软件版本控制等方面的内容。

当对软件进行维护时,软件产品发生了变化,这一系列的改变,必须在软件配置中体现出来,以防止因为维护所产生的变更给软件带来混乱。

## 习 题

1. 有哪几种类型的软件维护?

2. 某学校自己开发了一套计算机上机管理系统,学生可以通过上机卡刷卡上机,但是发现安全性不高,个别学生居然可以在没有刷卡的情况下跳过登录检控,而达到无卡上机的目的,因此,需要对系统进行改造。你认为这应该是一种什么类型的维护?

3. 某企业委托一家软件公司开发了一套工资报表生成系统。开始使用时还比较满意,但是随着工资数据的不断积累,报表生成速度越来越慢,以致月工资报表需要整整一天时间才能生成出来。因此,该企业要求软件公司对系统进行改造。你认为这应该是一种什么类型的维护?

4. 为什么系统越大越难维护?

5. 软件维护往往会对软件生产率带来负面影响,试对这一现象进行分析。

6. 什么是软件的可维护性?主要有哪些因素在影响着软件的可维护性。

7. 试对软件维护实施过程进行说明。

8. 什么是老化系统?对于老化系统的维护,Yourdon 提出了哪些建议?

9. 什么是逆向工程?什么是再工程?如何使用再工程对老化系统进行维护?

10. 什么是软件配置管理?软件维护中为什么需要特别关注软件配置管理?



# 附录 A 软件文档管理规范

## A.1 软件文档说明

### A.1.1 软件文档的定义及作用

软件文档是指某种数据媒体和其中所记录的数据。它具有永久性,并可以由人或机器阅读。正确地制作和使用软件文档,可以获得如下的便利:

1. 提高软件开发过程的能见度。
2. 提高开发效率。
3. 作为开发人员在一定阶段的工作成果和结束标志。
4. 记录开发过程中的有关信息,以便于协调以后的软件开发、使用和维护。
5. 便于潜在用户了解软件的功能、性能等各项指标,为他们选购符合自己需要的软件提供依据。

### A.1.2 软件文档分类

#### 1. 按照软件文档的形式进行分类

- 非正式文档:开发过程中需要经常填写的各种图表、工作表格。
- 正式文档:需要专门编制的各种技术资料、管理资料,一般要有文档编码号,并进行专门的存档管理。

其中,软件文档主要指正式文档。

#### 2. 按照软件文档的产生和使用范围进行分类

- 开发文档:软件开发过程中,作为软件开发人员前一阶段工作成果的体现和后一阶段工作依据的文档。包括可行性研究、项目开发计划、需求说明、数据说明、概要设计和详细设计。
- 管理文档:软件开发过程中,由软件开发人员制定的需提交管理人员的一些工作计划和工作报告,包括项目开发计划、测试计划、测试报告、开发进度月报及项目开发总结。
- 用户文档:软件开发人员为用户准备的有关该软件使用、操作、维护的资料,包括用户手册、操作手册、维护修改建议、需求说明。

#### 3. 按照国家标准分类

按照计算机软件产品开发文件编制指南的国家标准(GB 8567—88)的要求,在一项计算机软件的开发过程中,一般应产生以下文件:

- 可行性研究报告

- 项目开发计划
- 需求规格说明书
- 测试计划
- 概要设计说明书
- 数据库设计说明书
- 详细设计说明书
- 模块开发卷宗
- 用户操作手册
- 系统维护手册
- 测试分析报告
- 系统试运行计划
- 开发进度月报
- 项目开发总结报告

### A.1.3 软件文档与软件生命周期之间的关系

一般而言，软件生命周期可以分为六个阶段：

1. 项目可行性研究与规划阶段。
2. 软件需求分析阶段。
3. 软件设计阶段。
4. 软件实现阶段。
5. 软件测试阶段。
6. 软件运行与维护阶段。

在软件开发的阶段，需要产生不同的软件文档，具体对应关系如表 A-1 所列。

表 A-1 软件开发不同阶段需要产生的文档

阶段 文档	可行性研究与 规划阶段	需求分析 阶段	设计阶段	实现阶段	测试阶段	运行与维护 阶段
可行性研究报告						
项目开发计划						
需求规格说明书						
测试计划						
概要设计说明书						
数据库设计说明书						
详细设计说明书						
模块开发卷宗						
用户操作手册						
系统维护手册						
测试分析报告						
开发进度月报						
项目开发总结						

#### A.1.4 文档的使用者

对于软件文档的使用人员而言，与其所承担的工作有关，具体情况如下。

##### 1. 管理人员

- 可行性研究报告
- 项目开发计划
- 模块开发卷宗
- 开发进度月报
- 项目开发总结报告

##### 2. 开发人员

- 可行性研究报告
- 项目开发计划
- 软件需求说明书
- 数据要求说明书
- 详细设计说明书
- 数据库设计说明书

##### 3. 维护人员

- 设计说明书
- 测试分析报告
- 模块开发卷宗

##### 4. 最终用户

- 用户手册
- 操作手册

#### A.1.5 文档编码规则

为了实现对文档的规范管理，所有正式的软件文档都有必要编码号（用做文档的惟一标识），并需要有相应的文档编码规则与其配套。

文档编码要求能够体现出文档的基本特征，如开发机构标识、文件种类标识、创建年月、顺序号和版本号等。

例如下面列出的文档编码规则。

- 开发机构标识：由字符组成。  
例如：ABC。
- 文件种类标识：由 1 到 2 个字符组成。

例如：

R—需求分析报告

T—系统设计说明书

C—源代码

M—用户使用手册

- 创建年月号：由六位数字组成，前四位数字年份，后两位数字为月份。  
例如：200208，表示是 2002 年 8 月创建。
- 分类顺序号：由四位数字组成，用于每一类文件连续计数。
- 版本号：由一个数字组成，表示文件的更新次数。

假如某标识为 ABC 的公司，在 2002 年 6 月创建了“需求规格说明书”的第一版。则根据上述编码规则，可以得出该文档的编码号是：ABC-R-200206-0036-1。

需要注意的是：不同的软件开发机构都可以根据需要制定出适合自己机构特点的文档编码规则。

## A.2 软件文档格式

除非全部文档内容仅在一页以内，一般情况下软件文档的组成包括：封面、目录、版本更新说明、文件内容等，如图 A-1 所示。可以使用 Word 编辑文档。

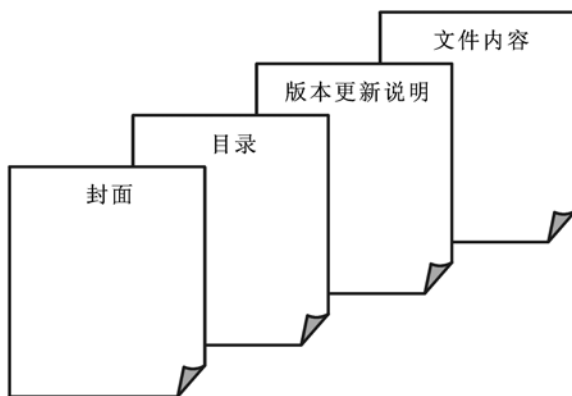


图 A-1 文档组成

### 1. 封面

文档封面需要体现的内容包括：

- 软件项目名称
- 文档名称
- 文档编码
- 保密级别
- 版本号
- 完成日期
- 作者姓名
- 软件机构名称

### 2. 目录

可采用手工编制或使用 Microsoft Word 的自动生成目录的功能产生文档目录。

### 3. 版本更新说明

当文档内容有更改时，如产生了文档的第二版、第三版时，需要加进版本更新说明。其内容包括：

- 更新日期
- 更新理由
- 更新人

### 4. 文件内容

文件内容主要成分是：

- 标题：文件内容须规定各个层次的标题，以方便目录的编制。各标题一般不使用标点符号，并按照科学编码法，从 1、1.1、1.1.1 开始编号。其中，一级标题使用四号加粗宋体，二级标题使用五号加粗宋体，三级标题使用五号普通宋体。例如下面的标题格式：

## 2. 可行性研究的前提

### 2.1 基本要求

#### 2.1.1 基本功能要求

- 正文：一般情况下，正文中的中文使用五号普通宋体，英文使用 Times New Roman 字体。
- 图表：图片不能跨页。表格可以跨页，但表格的第 2 页必须要有表头。图表需要编排顺序号，一般按一级标题编排，例如：图 1-1、表 1-1。
- 引用文献：文件中需要引用文献时，需要使用方框号“[ ]”引用，例如：参看文献[6, 7]。
- 术语：科技术语应采用国家标准（或行业标准）规定的（或通用的）术语或名称。对于新名词或特殊名词，需要在适当位置加以说明或注解。对于英文缩写词，应在文中第一次出现时用括号给出英文全文。注意文中名词术语的统一性。
- 参考文献：参考文献应按在文档中引用的先后次序排列。

著作文献编排格式是：

序号      作者      书名                  出版单位      出版年份      引用部分起止页

翻译文献编排格式是：

序号      作者      书名                  翻译者      出版单位      出版年份      引用部分起止页

- 附录：需要收录于文档中，但又不适合写进正文中的附加资料、公式推导等内容。若有多附录，可以采用序号“附录 A”、“附录 B”等。
- 索引：为了便于检索文中内容，可编制索引置于正文之后。索引以文档中的专业词语为关键字进行检索，指出其相关内容所在页码。中文词句索引按拼音排序，英文词句索引按英文字母排序。
- 页眉、页脚：一般页眉处标明一级标题名称，页脚处标明页码、日期，以保证单独的一页文档也能显示其正确属性。

## A.3 软件文档管理规则

为使软件文档管理规范，其需要有专门机构、专门人员负责，并需要有相应的管理规则、制度与其配套。

下面以某软件公司为例，对文档管理规则给予说明：

某软件公司为了承接较大型软件项目的开发，专门设立了软件开发部、软件测试中心和软件文档管理中心。软件开发任务由软件开发部承接，并针对具体软件任务组建专门的项目组具体实施。软件测试中心承担所有软件的测试任务。软件文档主要由软件项目组或软件测试中心创建，然后交软件开发部或软件测试中心经理审查，并由软件文档管理中心进行存档、复制与下发。

该软件开发机构组织结构如图 A-2 所示。

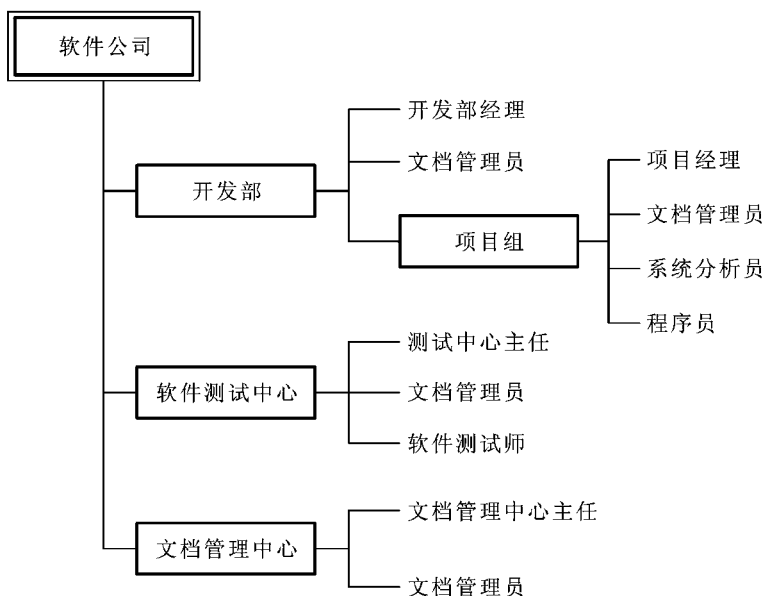


图 A-2 软件公司组织结构

为了使得开发部、软件测试中心和文档管理中心之间能够协调工作，使文档管理工作能够实际到位、落实到人，该公司专门制定了以下文档管理规则。

### 1. 开发部经理职责

- 制定项目任务书和项目管理文档
- 审批部门内部相关人员关于软件文档的制作、上报、借阅和复制的申请书

### 2. 开发部文档管理员职责

- 协助部门经理做好文档的制作、复制、记录和管理工作
- 下发管理文档给部门内部各个项目组
- 收集部门内部各个项目组的管理文档和技术文档

### 3. 项目经理职责

- 撰写《软件项目开发计划》
- 撰写《项目开发月报》

### 4. 项目组文档管理员职责

- 接受来自项目组成员的文档打字任务
- 根据文档制作规范，对文档的格式进行编辑、排版
- 进行文档的借阅、复制、打印等日常管理工作

### 5. 项目组系统分析员职责

- 撰写《需求规格说明书》

### 6. 项目组系统设计师职责

- 撰写《概要设计说明书》
- 撰写《详细设计说明书》

### 7. 项目组程序员职责

- 撰写《模块开发卷宗》
- 撰写《用户手册》
- 撰写《系统维护手册》

### 8. 软件测试中心主任职责

- 制定《项目测试管理计划书》
- 制定《项目集成测试计划书》
- 审查《项目集成测试报告书》
- 制定《项目单元测试计划书》
- 审查《项目单元测试报告书》

### 9. 软件测试中心文档管理员职责

- 接受来自测试中心的文档打字任务
- 根据文档制作规范，对文档的格式进行编辑、排版
- 进行测试文档的存档、借阅、复制、打印等日常管理工作

### 10. 软件测试中心软件测试师职责

- 撰写《项目集成测试报告书》
- 撰写《项目单元测试报告书》

### 11. 文档管理配置中心主任职责

- 审批管理文档和技术文档的借阅与复制
- 监督和检查关于管理文档和技术文档的配置管理质量

### 12. 文档管理配置中心管理员职责

- 制定《项目配置管理计划书》
- 定期收集或接受并记录来自开发部门项目组的管理文档和技术文档
- 建立关于特定项目的文档配置管理和版本控制体系
- 执行审批后的关于管理文档和技术文档的借阅与复制工作
- 对特定项目的管理文档和技术文档的质量进行评价

## A.4 软件文档的质量评价

高质量文档应该具有以下几个方面的特征：

1．规范性

符合文档制作规范。

2．针对性

编制文档时应当分清读者对象，并按不同类型、层次的读者特点，决定怎样适应他们的需要。

3．精确性

文档的行文应当十分准确，不能出现多义性的描述。

4．清晰性

文档编写应力求简明，并配以适当的图表，以增强其清晰性。

5．完整性

任何一个文档都应当是完整的、独立的，应该自成体系。

6．灵活性

不同软件项目，其规模、复杂程度都会有所差别，需要区别对待。



## 附录 B 软件文档格式

### B.1 可行性研究报告

#### 1. 引言

##### 1.1 编写目的

阐明编写可行性研究报告的目的，描述报告基本组成要素，指出读者对象。

##### 1.2 项目背景

列出拟开发软件系统的名称，项目委托者、开发者和预期用户，说明该软件系统与其他系统之间的关系等。

##### 1.3 定义

列出报告中所使用的专门术语的定义和缩写词的原意。

##### 1.4 参考资料

列出本文档需要引用的参考资料的来源。包括：已被批准的项目任务书、合同或上级机关批文，与项目有关的已经公开发表的论文、著作，需要采用的标准或规范。

#### 2. 可行性研究的前提

##### 2.1 基本要求

列出项目的各项基本要求，涉及功能、性能、数据输入、数据输出、数据处理流程、安全保密要求、与其他系统的关系、完成期限等。

##### 2.2 基本目标

涉及人力与设备费用的减少，工作环境的改善，工作效率的提高，处理速度、控制精度或生产能力的提高，信息管理服务的改进，自动决策功能的改进等。

##### 2.3 条件、假定和限制

涉及拟开发系统运行寿命的最小值，项目经费来源与限制，政策和法规方面的限制，硬件、软件、开发环境和运行环境方面的条件与限制，可以利用的相关资源，拟开发系统要求投入使用的最迟时间等。

#### 3. 对现有系统的分析

##### 3.1 系统模型

可使用系统方框图、系统流程图说明现有系统的基本构造与基本处理流程。

##### 3.2 工作负荷

列出现有系统所承担的工作与工作量。

##### 3.3 费用支出

涉及现有系统运行过程中的人力、设备、空间、支持性服务和材料等各项支出。

##### 3.4 局限性

指出现有系统存在的问题，并说明开发新系统或改造现有系统的必要性。

#### 4. 对拟开发系统的分析

##### 4.1 拟开发系统的体系结构

可使用系统方框图对拟开发系统的体系结构进行概要描述。

##### 4.2 拟开发系统的工作模型

可使用系统流程图说明拟开发系统的基本处理流程。

##### 4.3 拟开发系统的优越性

将拟开发系统与现有系统进行对比，并在诸如提高处理能力、减轻工作负荷、增强系统灵活性和保证数据安全等方面，说明拟开发系统所具有的优越性。

##### 4.4 拟开发系统可能带来的影响

涉及拟开发系统将对硬件设备、软件配置和用户操作等方面带来的影响。

#### 5. 对拟开发系统的可行性评价

##### 5.1 技术可行性评价

说明拟开发系统在技术方面具备的可行性，例如，在当前技术允许的条件下，该系统的功能目标能否达到；在规定的时间内，该系统的开发能否按期完工。

##### 5.2 经济可行性评价

说明拟开发系统所需的开发费用和可以预期的经济收益，并由此进行成本效益分析。

##### 5.3 社会因素的可行性评价

从法律法规、用户操作规程等方面进行可行性评价，例如项目合同中责任是否明确，拟开发系统是否存在著作权侵权，是否侵犯了国家、集体或他人的利益，拟开发系统是否充分考虑到了用户的组织管理结构、工作流程、人员素质诸多方面的因素等。

#### 6. 其他可供选择的方案

扼要说明其他可供选择的方案，并说明未被选取的理由。

#### 7. 可行性分析结论

几种可能的结论：

(1) 该项目各项条件已经具备，建议立即着手组织实施。

(2) 建议等待项目需要的有关条件成熟后，例如人力、设备和资金等到位之后，才开始组织实施。

(3) 该项目基本条件不够成熟，例如关键技术难以突破，资金缺口太大，经济上不合算，因此项目不能进行或不必要进行下去。

## B.2 项目计划说明书

### 1. 引言

#### 1.1 编写目的

阐明编写项目计划说明书目的，指出读者对象。

#### 1.2 项目背景

列出本项目的任务提出者、开发机构和主管部门，说明待开发软件系统与其他系统之间的关系。

### 1.3 定义

列出报告中所使用的专门术语的定义和缩写词的原意。

### 1.4 参考资料

列出本文档需要引用的参考资料的来源。包括已被批准的项目任务书、合同或上级机关批文，被引用的资料、标准或规范。

## 2 项目概述

### 2.1 工作内容

说明各项工作的主要内容，简要介绍所开发软件的功能、性能等。

### 2.2 条件与限制

列出影响开发人员在设计软件时的约束条款，例如完成项目应具备的条件，开发机构已具备的条件，尚需创造的条件，用户及项目合同承包者需要承担的责任，项目完工期限等。

### 2.3 产品

#### 2.3.1 程序

列出需移交给用户的程序的名称、所用的编程语言及程序的存储形式等。

#### 2.3.2 文档

列出需移交给用户的每种文档的名称及内容要点。

### 2.4 运行环境

描述软件的运行环境，包括硬件平台、操作系统以及其他的软件组件或与其协同工作的其他应用程序等。

### 2.5 服务内容

列出开发单位需向用户提供的各项服务，包括人员培训、安装、保修、维护以及其他运行支持等，并逐项规定开始日期、所提供支持的级别和服务的期限。

### 2.6 验收标准

简要说明对所开发的软件产品进行用户确认的验收标准及其依据。

## 3 实施计划

### 3.1 任务分解

描述任务的划分情况，并列出各项任务的负责人。

### 3.2 进度

按阶段分解项目任务，例如需求分析、设计、编码实现、测试、移交、培训和安装等。可使用图表（例如甘特图）安排各项任务，涉及任务的开始时间、完成时间、先后顺序和所需资源等。

### 3.3 预算

列出本项目所需要的劳务（包括人员的数量和时间）、经费的预算（包括办公费、差旅费、机时费、资料费、通信设备和专用设备的租金等）和来源。

### 3.4 关键问题

指出能够影响整个项目成败的关键问题、技术难点和风险。说明这些问题对项目的影响及解决方法、途径。

## 4 人员组织及分工

扼要说明参加本项目开发工作的主要人员的基本情况，包括职务、学历、职称、职位、技

术特长和作用等。

#### 5. 交付期限

给出项目完工并交付使用的预期期限。

#### 6. 专题计划要点

扼要说明项目实施过程中需制定的各个专题计划，例如开发人员培训计划、测试计划、安全保密计划、质量保证计划、配置管理计划、用户培训计划和系统安装计划。

## B.3 需求规格说明书

### 1. 引言

#### 1.1 编写目的

阐明编写本需求规格说明书的目的，指出读者对象。

#### 1.2 项目背景

列出本项目的委托单位、开发单位和主管部门，说明该软件系统与其他系统的关系。

#### 1.3 定义

列出本文档中所用到的专门术语的定义和缩写词的原意。

#### 1.4 参考资料

列出本文档需要参考的有关资料的作者、标题、编号、发表日期、出版单位或资料来源。包括本项目经核准的计划任务书、合同或上级机关的批文，项目开发计划，本文档需要引用的论文、著作，需要采用的标准、规范。

### 2. 系统概述

#### 2.1 系统定义

说明系统目标，定义系统边界，描述系统业务规则。可使用顶层数据流程图或系统用例图进行描述。

#### 2.2 处理流程

可使用数据流程图或系统活动图描述系统的处理流程。

#### 2.3 运行环境

说明系统运行时需要具备的硬件环境、操作系统、支撑软件环境、数据环境、网络环境以及需要一起协同工作的其他软件和设备环境等。

#### 2.4 条件与限制

包括对编程语言、数据库管理系统的限制，硬件设备的限制，所要求的开发规范或标准的限制等。

### 3. 功能需求

#### 3.1 功能划分

可使用系统层次图说明系统的功能组织结构。

#### 3.2 功能描述

对系统中的每一项功能进行细节说明。

### 4. 性能需求

#### 4.1 数据精确度

输出数据的精确位数。

#### 4.2 时间特性

例如操作响应时间、更新处理时间、数据转换与传输时间、运行时间等。

#### 4.3 适应性

当操作方式、运行环境、与其他软件的接口以及开发计划等发生变化时，系统应具有适应能力。

### 5. 运行需求

#### 5.1 用户界面

例如屏幕格式、报表格式、菜单格式等。

#### 5.2 硬件接口

说明系统需要依赖的硬件接口（例如串行接口、并行接口、USB 接口）和硬件接口能够支持的设备。

#### 5.3 软件接口

说明系统需要依赖的其他软件系统的接口。

#### 5.4 通信接口

说明系统需要依赖的各种通信需求，例如网络设备、网络通讯协议、电子邮件、Web 浏览器等。

#### 5.5 故障处理

说明对各种可能的软件、硬件故障将要进行的处理。

### 6. 其他需求

涉及可使用性、安全保密性、可维护性、可移植性等。

### 7. 数据描述

#### 7.1 静态数据

可使用 ER 图建立数据实体概念模型，以说明静态数据结构。

#### 7.2 动态数据

包括输入数据和输出数据。

#### 7.3 数据库描述

给出需要使用的数据库的名称和类型。

#### 7.4 数据词典

需要对图形模型中的图形元素逐个给出词条解释。

#### 7.5 数据采集

说明数据采集对象与方法，包括提供输入数据的机构、设备和人员，输入数据的手段、介质和设备，数据生成的方法、介质和设备等。

## B.4 概要设计说明书

### 1. 引言

### 1.1 编写目的

阐明编写本概要设计说明书的目的，指出读者对象。

### 1.2 项目背景

列出本项目的委托单位、开发单位和主管部门，说明该软件系统与其他系统的关系。

### 1.3 定义

列出本文档中所用到的专门术语的定义和缩写词的原意。

### 1.4 参考资料

列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源。包括本项目经核准的计划任务书、合同或上级机关的批文，项目开发计划，需求规格说明书，本文档需要引用的论文、著作，需要采用的标准、规范。

## 2. 需求概述

### 2.1 功能要求

列出必须实现的功能，可以扩充的功能。可使用系统层次图说明系统功能组织结构。

### 2.2 性能要求

说明系统需要具备数据精度、时间特性、适应性等。

### 2.3 运行环境

说明系统运行时需要具备的硬件环境、操作系统、支撑软件环境、数据环境、网络环境以及需要一起协同工作的其他软件和设备环境等。

### 2.4 条件与限制

说明软件系统在功能和性能方面的特殊要求。

## 3. 系统设计目标

说明概要设计将要采用的设计思路 and 需要达到的设计目标。

## 4. 系统设计原则

说明概要设计需要遵循的设计原则。

## 5. 系统总体设计

### 5.1 处理流程

使用系统流程图说明软件系统的处理流程。

### 5.2 总体结构设计

描述软件系统中模块的组成关系。可使用软件结构图或类图进行描述。

### 5.3 功能分配

说明各项功能与软件结构的关系。

## 6. 模块接口设计

### 6.1 外部接口

说明系统与其他外部组件之间的接口关系，包括界面接口、软件接口与硬件接口。

### 6.2 内部接口

说明系统内部各个模块之间的接口、调用关系以及模块之间的数据传递关系等。

## 7. 数据结构设计

### 7.1 逻辑结构设计

描述系统需要使用的数据（如数据表、数据项、记录、文件）的标识、定义、长度等逻辑特征以及它们之间的逻辑关系。

## 7.2 物理结构设计

描述系统需要使用的数据（如数据表、数据项、记录、文件）的存储要求、访问方式、存取单位等物理特征以及它们之间的物理关系。

## 7.3 数据结构与程序的关系

描述系统需要使用的数据与程序模块之间的关联，说明哪些数据将被哪些模块所用。

## 8. 运行控制设计

### 8.1 运行模块的组合

描述系统运行时，模块之间的通信与组合关系。可使用模块结构图、对象协作图等进行描述。

### 8.2 运行控制

描述系统运行时，模块之间的调用控制关系，涉及控制范围、作用范围等。

### 8.3 运行时间

说明系统运行时，系统中的运行模块对时间的要求。

## 9. 出错处理设计

### 9.1 出错输出信息

用图表方式列出当每种可能的出错或故障情况出现时，系统出错信息的输出形式、含义及处理方法。

### 9.2 出错处理对策

如设置后备服务、性能降级、恢复及再启动等。

## 10. 系统安全性保密性设计

说明对人员登录、操作历史、数据编辑等方面的保密和控制措施。

### 10.1 系统操作权限分级管理

将系统操作权限分为系统管理员级、部门级、岗位级和个人级，采用口令方式登录，口令可由使用者自行设置。

### 10.2 特定功能的操作校验

为保证数据读写、更改、删除无误，可以采用警告信息提请注意，经确认后再提交上述编辑操作。

### 10.3 文件与数据加密

可利用数据库管理系统或前台开发工具所具有的安全保密功能，对文件和数据进行加密处理。

### 10.4 非法使用数据的记录和检查

可建立操作日志，对每一个操作点的操作内容进行全程自动记录。系统内保存至少半个月的日志记录以备查，采用光盘或磁带备份一年内的操作日志记录。

## 11. 系统维护设计

说明为了系统维护便利，需要在程序设计中采取的策略，例如可在程序中专门安排用于系统的检查与维护的检测点和专用模块。

## B.5 数据库设计说明书

### 1. 引言

#### 1.1 编写目的

阐明编写本数据库设计说明书的目的，指出读者对象。

#### 1.2 项目背景

列出本项目的委托单位、开发单位和主管部门，说明该数据库系统与其他系统的关系。

#### 1.3 定义

列出本文档中所用到的专门术语的定义和缩写词的原意。

#### 1.4 参考资料

列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源。包括本项目经核准的计划任务书、合同或上级机关的批文，项目开发计划，需求规格说明书，本文档需要引用的论文、著作，需要采用的标准、规范。

### 2. 外部设计

#### 2.1 标识

列出用于标识该数据库的编码、名称、标识符或标号，并给出附加的描述性信息。如果该数据库是在实验中的或是暂时性的，则要说明其暂时性和有效期。

#### 2.2 约定

叙述使用该数据库所必须了解的建立标号、标识的有关约定。例如用于标识库内各个文卷、记录、数据项的命名约定等。

#### 2.3 使用该数据库的软件

列出将要使用或访问该数据库的所有软件。

#### 2.4 支撑软件

叙述与此数据库有关的支撑软件，如数据库管理系统、存储定位程序等。概要说明这些支撑软件的名称、功能及为使用这些支撑软件所需的操作命令。列出这些支撑软件的有关资料。

#### 2.5 专门说明

为此数据库的生成、测试、操作和维护的相关人员提供专门的说明。

### 3. 结构设计

#### 3.1 概念结构设计

说明数据库的用户视图，即反映现实世界中的实体、属性和它们之间关系的原始数据形式，包括各数据项、记录、文卷的标识符、定义、类型、度量单位和值域。可使用 ER 图。

#### 3.2 逻辑结构设计

说明把上述原始数据进行分解、合并后重新组织起来的数据库全局逻辑结构，包括记录、段的编排，记录、段之间的关系及存取方法等，形成本数据库的管理员视图。

#### 3.3 物理结构设计

建立系统程序员视图，包括：



(1) 数据在内存中的安排, 包括索引区、缓冲区的设计。

(2) 所使用的外存设备及外存之间的组织, 包括索引区、数据块的组织与划分。

(3) 访问数据的方式方法。

#### 4. 运用设计

##### 4.1 数据字典设计

对数据库设计中涉及的数据项、记录、文卷、子模式、模式等一般要建立数据字典, 以说明它们的标识符、同义名及有关信息。

##### 4.2 完整性设计

说明为保持数据库中数据的完整性所作的考虑, 如数据库的后援频率、数据共享、数据冗余等。

##### 4.3 完全保密设计

说明所采用的保证数据安全保密的措施和机制, 如数据库安全破坏标识、资源保护方式、存取控制方式等。

## B.6 详细设计说明书

### 1. 引言

#### 1.1 编写目的

阐明编写本详细设计说明书的目的, 指出读者对象。

#### 1.2 项目背景

列出本项目的委托单位、开发单位和主管部门, 说明该软件系统与其他系统的关系。

#### 1.3 定义

列出本文档中所用到的专门术语的定义和缩写词的原意。

#### 1.4 参考资料

列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源。包括本项目经核准的计划任务书、合同或上级机关的批文, 项目开发计划, 需求规格说明书, 概要设计说明书, 数据库设计说明书, 本文档中所引用的资料、采用的标准和规范。

### 2. 总体设计概述

#### 2.1 需求概述

给出关于系统需求方面的必要说明。

#### 2.2 软件结构

可使用软件结构图形式给出系统概要设计中关于软件模块结构及相互关系的说明。

### 3. 程序描述

需要逐个模块给出以下说明。

#### 3.1 功能

说明该模块的功能设计。

#### 3.2 性能

说明该模块的性能设计。

### 3.3 输入项目

说明该模块的输入参数的名称、数据类型、数目、先后顺序。

### 3.4 输出项目

说明该模块的输出参数的名称、数据类型、数目、先后顺序。

### 3.5 算法

说明该模块所选用的算法和流程。

### 3.6 程序逻辑

详细描述模块实现的算法，可采用标准流程图、PDL 语言、NS 图、PAD 图、判定表等描述算法。

### 3.7 接口

使用图表反映本模块的上级模块、下级模块，说明模块之间的调用关系、参数赋值，以及与本模块有关的外部数据源等。

### 3.8 存储分配

说明该模块在设计时和运行时的物理的和逻辑的绝对位置和相对位置。

### 3.9 限制条件

说明该模块正常工作的必需条件以及该模块功能的适用范围。

### 3.10 测试要点

给出测试本模块的主要测试要求，如对测试技术、输入数据、预期结果的规定。

## B.7 模块开发卷宗

### 1. 引言

#### 1.1 编写目的

阐明编写本系统模块开发卷宗的目的，指出读者对象。

#### 1.2 项目背景

列出本项目的委托单位、开发单位和主管部门，说明该软件系统与其他系统的关系。

#### 1.3 定义

列出本文档中所用到的专门术语的定义和缩写词的原意。

#### 1.4 参考资料

列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源。包括本项目经核准的计划任务书、合同或上级机关的批文，项目开发计划，需求规格说明书，概要设计说明书，数据库设计说明书，详细设计说明书，本文档中所引用的资料、采用的标准和规范。

### 2. 修改情况说明

包括：软件系统名称和标识符、模块名称和标识符、卷宗序号、修改文本序号、修改完成日期、编排日期等。

### 3. 模块开发情况表

可使用下表记录模块开发情况：

模块标识符		
模块的描述性名称		
代码设计	计划开始日期	
	实际开始日期	
	计划完成日期	
	实际完成日期	
模块测试	计划开始日期	
	实际开始日期	
	计划完成日期	
	实际完成日期	
组装测试	计划开始日期	
	实际开始日期	
	计划完成日期	
	实际完成日期	
代码复查日期、签字		
源代码行数	预 计	
	实 际	
目标模块大小	预 计	
	实 际	
项目负责人批准日期、签字		

#### 4. 功能说明

扼要说明本模块的功能，主要是输入、要求的处理、输出。同时列出在软件需求说明书中对这些功能的说明的目录序号。

#### 5. 设计说明

说明本模块的设计考虑：在设计说明书中有关对本模块设计考虑的叙述，包括本模块在软件系统中所处的层次，同其他模块的接口等。其中，在设计说明书中有关对本模块的设计考虑，包括本模块的算法、处理流程、牵涉到的数据文卷设计限制、驱动方式和出错信息等以及目前已通过测试的源代码对设计的实际使用情况的说明等。

#### 6. 源代码清单

给出已通过测试的本模块无语法错误的源代码清单。

#### 7. 测试说明

说明需要针对本模块的每一项测试，包括测试标识号、测试目的、所用的配置、输入和预期的输出及实际的输出等。

#### 8. 复审结论

可把实际测试的结果，同软件需求规格说明书、概要设计说明书、详细设计说明书中规定的要求进行比较，然后得出结论。

## B.8 用户操作手册

### 1. 引言

### 1.1 编写目的

阐明编写本系统维护手册的目的，指出读者对象。

### 1.2 项目背景

列出本项目的委托单位、开发单位和主管部门，说明该软件系统与其他系统的关系。

### 1.3 定义

列出本文档中所用到的专门术语的定义和缩写词的原意。

### 1.4 参考资料

列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源。包括本项目经核准的计划任务书、合同或上级机关的批文，项目开发计划，需求规格说明书，概要设计说明书，数据库设计说明书，详细设计说明书，模块开发卷宗，本文档中所引用的资料、采用的标准和规范。

## 2. 软件概述

### 2.1 功能

逐项说明本软件所具有的功能。

### 2.2 性能

#### 2.2.1 数据精确度

逐项说明各个输入数据需要的精度与输出数据可以达到的精度。

#### 2.2.2 时间特性

定量说明软件的时间精度，如响应时间、处理时间、数据传输时间等。

#### 2.2.3 适应性

说明当用户在操作方式、运行环境方面需作某些变更时，本软件所具有的适应能力。

#### 2.2.4 安全保密

说明软件在安全保密方面所做的设计考虑与所具有的能力。

#### 2.2.5 数据库

说明本软件与所依赖的数据库之间的关系。

## 3. 运行环境

### 3.1 硬件环境

列出本软件系统运行时所需要的硬件设备及其最小配置，包括计算机型号、主存容量，外存储设备，输入、输出设备，数据传输设备，数据转换设备等。

### 3.2 软件环境

包括操作系统名称及版本号，语言编译系统或汇编系统的名称及版本号，数据库管理系统的名称及版本号，其他必要的支持软件。

## 4. 使用说明

### 4.1 安装和初始化

给出程序的存储形式、操作命令、反馈信息及其含意，说明安装与初始化的过程实例以及安装所需的软件工具等。

### 4.2 输入

给出每项输入数据的说明。

#### 4.2.1 输入数据背景

说明输入数据的来源、存储媒体、出现频度、限制及合理性检测规则等。

#### 4.2.2 输入数据格式

包括数据的长度、格式基准、标号、顺序、分隔符，以及控制、省略、重复等。

#### 4.2.3 输入举例

为每个输入数据项提供操作实例。

### 4.3 输出

给出每项输出数据的说明。

#### 4.3.1 数据背景

说明输出数据的去向、使用频度、存放媒体及合理性检测规则等。

#### 4.3.2 数据格式

说明每一个输出数据的格式，如首部、主体和尾部的具体形式。

#### 4.3.3 举例

为每个输出数据项提供操作实例。

### 4.4 出错和恢复

给出本软件的出错信息编码及其含意，说明用户应承担的纠错任务及其措施，如修改、恢复、再启动。

### 4.5 求助查询

说明如何获得操作提示或联机帮助信息。

## 5. 运行说明

### 5.1 运行表

列出每种可能的运行情况，说明其运行目的。

### 5.2 运行步骤

按顺序说明每种运行的操作步骤。

### 5.3 输入、输出文件

给出需要建立或更新的文件的有关信息，如文件的名称及编号、记录媒体、存留的目录以及对文件的支配方法与要求等。

## 6. 非常规过程

给出应急或非常规操作的必要信息及操作步骤，如出错处理操作、向后备系统的切换操作以及维护人员须知的操作和注意事项等。

## 7. 操作命令一览表

按字母顺序逐个列出全部操作命令的格式、功能及参数说明。

## 8. 程序文件（或命令文件）和数据文件一览表

按文件名字母顺序或按功能分类顺序逐个列出文件名称、标识符及说明。

## 9. 用户操作举例

提供一些有关用户操作的典型实例。

## B.9 系统维护手册

### 1. 引言

### 1.1 编写目的

阐明编写本系统维护手册的目的，指出读者对象。

### 1.2 项目背景

列出本项目的委托单位、开发单位和主管部门；该软件系统与其他系统的关系。

### 1.3 定义

列出本文档中所用到的专门术语的定义和缩写词的原意。

### 1.4 参考资料

列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源。包括本项目经核准的计划任务书、合同或上级机关的批文，项目开发计划，需求规格说明书，概要设计说明书，数据库设计说明书，详细设计说明书，模块开发卷宗，本文档中所引用的资料、采用的标准和规范。

## 2. 系统说明

### 2.1 系统用途

说明系统具备的功能、输入和输出。

### 2.2 安全保密

说明系统安全保密方面的考虑。

### 2.3 总体说明

说明系统的总体功能，对系统、子系统和作业等作出综合性的介绍，并用图表方式给出系统主要部分的内部关系。

### 2.4 程序说明

说明系统中每一程序、分程序的细节和特性。主要包括以下内容：

- (1) 程序所具有的功能
- (2) 程序功能实现方法
- (3) 程序中的输入数据
- (4) 程序中对数据的处理流程
- (5) 程序中的输出数据
- (6) 系统内部程序之间的接口
- (7) 程序中需要使用的各种表格

## 3. 操作环境

### 3.1 设备

逐项说明系统的设备配置及其特性。

### 3.2 支持软件

列出系统使用的支持软件，包括它们的名称和版本号。

### 3.3 数据库

#### 3.3.1 总体特征

包括数据库标识符，使用这些数据库的程序，静态数据、动态数据，数据库的存储媒体，程序使用数据库的限制等。

#### 3.3.2 结构及详细说明

说明数据库的结构，涉及数据库中的表、记录、字段等。

#### 4. 维护过程

##### 4.1 约定

列出该软件设计中所使用的全部规则和约定。

##### 4.2 验证过程

说明一个程序段修改后，对其进行验证的要求和过程。

##### 4.3 出错及纠正方法

列出出错状态及其纠正方法。

##### 4.4 专门维护过程

说明本文档其他地方没有提到的专门维护过程。

##### 4.5 专用维护程序

列出维护软件系统使用的后备技术和专用程序，如文件恢复程序。

##### 4.6 程序清单和流程图

引用资料或提供附录给出程序清单和流程图。

## B.10 测试计划书

### 1. 引言

#### 1.1 编写目的

阐明编写本系统测试计划书的目的，指出读者对象。

#### 1.2 项目背景

列出本项目的委托单位、开发单位和主管部门，说明该软件系统与其他系统的关系。

#### 1.3 定义

列出本文档中所用到的专门术语的定义和缩写词的原意。

#### 1.4 参考资料

列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源。包括本项目经核准的计划任务书、合同或上级机关的批文，项目开发计划，需求规格说明书，概要设计说明书，数据库设计说明书，详细设计说明书，本文档中所引用的资料、采用的标准和规范。

### 2. 需求概述

#### 2.1 功能

可使用系统层次图说明系统功能组织结构。

#### 2.2 性能

说明系统需要具备数据精度、时间特性、适应性等。

#### 2.3 运行环境

说明系统运行时需要具备的硬件环境、操作系统、支撑软件环境、数据环境、网络环境以及需要一起协同工作的其他软件和设备环境等。

#### 2.4 条件与限制

说明软件系统在功能和性能方面的特殊要求。

### 3. 测试计划

### 3.1 测试方案

说明确定测试方法和选取测试用例的原则。

### 3.2 测试项目与进度

列出测试计划中需要进行的每一项测试的内容、名称、目的和时间进度。

### 3.3 测试机构

列出测试机构名称、负责人和职责。

## 4. 评价标准

### 4.1 测试范围

说明需要进行的各项测试的检测范围及其局限性。

### 4.2 测试数据转换方法

说明将原始测试数据转换成能够便于评价的数据形式的转换技术与方法。

### 4.3 准则

说明评价测试结果时将要采用的标准，例如出错概率最大限值、时间响应最大限值。

## 5. 测试项目说明

按顺序逐个对测试项目做出说明。

### 5.1 测试项目名称

#### 5.1.1 测试条件

给出该项测试对资源的特殊要求，包括设备配置、软件配置、人员安排与培训以及测试资料等。

#### 5.1.2 测试用例设计

可使用表格将该项测试需要的用例逐个列出，包括用例标识、输入数据、输入命令、预期的输出结果、操作步骤和允许的偏差等。

#### 5.1.3 测试结果评价

可用表格将该项测试需要给出的评价逐个列出，例如下面表格中关于程序模块的编码质量测试评价。

测 评 项 目	测 评 结 果	修 改 意 见
程序风格	好    一般    差	
输入/输出	好    一般    差	
模块命名	好    一般    差	
全局变量	好    一般    差	
局部变量	好    一般    差	
语法	好    一般    差	
结构化程度	好    一般    差	
功能目标	好    一般    差	

## B.11 测试分析报告

### 1. 引言

#### 1.1 编写目的



阐明编写测试分析报告的目的，指明读者对象。

### 1.2 项目背景

说明项目的来源、委托单位及主管部门。

### 1.3 定义

列出测试分析报告中用到的专门术语的定义和缩写词的原意。

### 1.4 参考资料

列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源。包括项目的计划任务书、合同或批文，项目开发计划，需求规格说明书，概要设计说明书，详细设计说明书，用户操作手册，测试计划，以及测试分析报告所引用的其他资料、采用的工程标准或规范。

## 2. 测试计划执行情况

### 2.1 测试项目

列出每一测试项目的名称、内容和目的。

### 2.2 测试机构和人员

给出测试机构名称、负责人和参与测试人员名单。

### 2.3 测试结果

可采用表格形式列出每一测试项目的实测数据、与预期结果的偏差、该项测试表明的事实和该项测试发现的问题。

## 3. 软件需求测试结论

列出每一项需求测试结论，包括能够证实的软件能力、软件的局限性。

## 4. 软件能力评价

### 4.1 软件能力

经过测试所表明的软件能力。

### 4.2 缺陷和限制

说明测试所揭露的软件缺陷和不足以及可能给软件运行带来的影响。

### 4.3 建议

提出为弥补上述缺陷的建议。

## 5. 测试分析结论

说明能否通过。

## B.12 系统试运行计划书

### 1. 引言

#### 1.1 编写目的

阐明编写本系统试运行计划书的目的，指出读者对象。

#### 1.2 项目背景

列出本项目的委托单位、开发单位和主管部门，说明该软件系统与其他系统的关系。

#### 1.3 定义

列出本文档中所用到的专门术语的定义和缩写词的原意。

### 1.4 参考资料

列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源。包括本项目经核准的计划任务书、合同或上级机关的批文，项目开发计划，需求规格说明书，概要设计说明书，数据库设计说明书，详细设计说明书，模块开发卷宗，系统测试报告，系统维护手册，用户手册，本文档中所引用的资料、采用的标准和规范。

### 2. 目标

说明应用系统安排试运行阶段的总体目标。

### 3. 进度安排

针对试运行阶段的人员、任务、地点和时间进度等作出具体安排。

### 4. 环境准备

针对试运行阶段必须具备的网络和软硬件环境作出明细说明。

### 5. 系统安装

针对系统功能构件组成及安装的方式方法作出详尽的描述。

### 6. 试运行执行记录

针对试运行阶段的各种操作细节加以表格化专人记录。

### 7. 试运行总结报告

在试运行阶段结束后，必需针对试运行的记录情况加以整理和归纳，出具试运行阶段的总结报告。

## B.13 项目开发总结报告

### 1. 引言

#### 1.1 编写目的

阐明编写本系统试运行计划书的目的，指出读者对象。

#### 1.2 项目背景

列出本项目的委托单位、开发单位和主管部门。

#### 1.3 定义

列出本文档中所用到的专门术语的定义和缩写词的原意。

#### 1.4 参考资料

列出有关资料的作者、标题、编号、发表日期、出版单位或资料来源。可包括：本项目经核准的计划任务书、合同或上级机关的批文，项目开发计划，需求规格说明书，概要设计说明书，数据库设计说明书，详细设计说明书，模块开发卷宗，系统测试报告，系统维护手册，用户手册，本文档中所引用的资料、采用的标准和规范。

### 2. 开发结果

#### 2.1 产品

列出最终形成的产品，包括：

(1) 程序名称、源程序行数（或目标程序字节数）、存储形式

(2) 软件系统版本号

### (3) 文档名称

### (4) 数据库名称

#### 2.2 主要功能及性能

列出本软件系统所实际具有的主要功能和性能，对照可行性研究报告、项目开发计划、需求说明书，说明原定的目标是达到了，还是未完全达到或超过了。

#### 2.3 基本流程

用图给出本软件系统实际的处理流程。

#### 2.4 进度

列出实际进度并与原定计划对比，指出实际进度比计划进度是提前还是延迟，并分析其主要原因。

#### 2.5 费用

列出实际支出费用并与原定计划费用对比，包括工时、计算机及其他设备使用时间，以及差旅费、印刷费、培训费等其他物料消耗费用。说明费用计划执行情况，并分析超支或节余的主要原因。

### 3. 开发工作评价

#### 3.1 生产效率的评价

包括程序的平均生产效率和文档的平均生产效率，并与原定计划进行对比。

#### 3.2 产品质量的评价

说明在测试阶段检查出来的程序的错误发生率，即每千条语句中的错误语句数。

#### 3.3 技术方法的评价

对开发工作中所使用的技术、方法、工具、手段作出评价。

### 4. 经验与教训

列出从这项开发工作中所得到的最主要的经验与教训，并对今后开发工作提出建议。

## 参 考 文 献

- 1 Ian Sommerville 著, 程成, 陈霞译. 软件工程. 第 6 版. 北京: 机械工业出版社, 2003
- 2 郑人杰, 殷人昆, 陶永雷. 实用软件工程. 第 2 版. 北京: 清华大学出版社, 1997
- 3 Ivar Jacobson, Grady Booch, James Rumbaugh 著. 周伯生等译. 统一软件开发过程. 北京: 机械工业出版社, 2002
- 4 Grady Booch, James Rumbaugh, Ivar Jacobson 著. 邵维忠等译. UML 用户指南. 北京: 机械工业出版社, 2001
- 5 Leszek A. Maciaszek 著. 金芝译. 需求分析与系统设计. 北京: 机械工业出版社, 中信出版社, 2003
- 6 张海藩编著. 软件工程导论. 第 4 版. 北京: 清华大学出版社, 2003
- 7 Edward Yourdon, Carl Argila 著. 殷人昆等译. 实用面向对象软件工程教程. 北京: 电子工业出版社, 1998
- 8 Karl E. Wiegers 著. 陆丽娜等译. 软件需求. 北京: 机械工业出版社, 2000
- 9 曾强聪编著. Visual Basic 程序设计与应用开发案例教程. 北京: 清华大学出版社, 2004